TREBALL FINAL DE GRAU

TÍTOL: IMPLEMENTACIÓ D'UN GENERADOR PROCEDURAL DE PLANETES

AUTORS: FERNÁNDEZ, BRIONES, ORIOL

DATA DE PRESENTACIÓ: JULIOL, 2022

COGNOMS: FERNÀNDEZ BRIONES **NOM: ORIOL**

TITULACIÓ: GRAU EN ENGINYERIA INFORMÀTICA

PLA: 2018

DIRECTOR: MANEL MORENO LUPIAÑEZ

DEPARTAMENT: FÍSICA

QUALIFICACIÓ DEL TFG

TRIBUNAL

PRESIDENT **SECRETARI** **VOCAL**

DATA DE LECTURA:

Aquest Projecte té en compte aspectes mediambientals: □ Sí □ No

RESUM

Aquest treball consisteix en l'estudi i desenvolupament d'una aplicació web per visualitzar l'evolució temporal de planetes com la Terra mitjançant gràfics per ordinador.

En cada capítol del document es presenta un nou concepte o característica que en conjunt, conformen un model simplificat amb l'objectiu de recrear el desenvolupament que va experimentar planeta, des de la seva formació fins a l'actualitat, explorant els diferents mecanismes que l'han portat a ser com el coneixem avui dia.

Alguns d'aquests són el naixement del protoplaneta en el període de formació inicial, l'origen de l'aigua, oceans i rius i els efectes erosius que causen en el terreny, la formació i moviment de les plaques tectòniques, els efectes de la llum en travessar una atmosfera i dispersar-se i el funcionament del clima a escala global entre altres coses. Com s'explica al document, simular tots aquests fenòmens és una tasca molt complexa i s'han realitzat moltes simplificacions per tal de fer viable aquest projecte.

L'objectiu final no és tractar de simular el planeta Terra, ja que és una tasca de gran complexitat i amb infinitat de variables, en comptes d'això, es vol explorar la idea d'implementar de forma simplificada el que coneixem del nostre planeta, i també d'altres, per a obtenir resultats aproximats que puguem extrapolar per a poder veure quin aspecte podrien tindre altres planetes habitables com el nostre, o crear uns altres totalment únics amb característiques molt diverses.

Paraules clau:

Simulador gràfic	Ombrejador (Shaders)	Gràfics per ordinador	Planetes terrestres
Evolució planetària	Tectònica de plaques	Erosió hidràulica	Atmosfera
OpenGL	Clima		

ABSTRACT

This work consists of the study and development of a web application to visualize the temporal evolution of planets such as Earth using computer graphics.

Each chapter of the document introduces a new concept or functionality which together form a simplified model in order to recreate the development experienced by the planet, from its formation to the present day, by exploring the various mechanisms that led it to be as we know it today.

Some of these are the birth of the protoplanet in the initial formation period, the origin of water, oceans and rivers and the erosive effects they produce on the soil, the formation and movement of tectonic plates, the effects of the light as it traverses an atmosphere and disperses and how climate works on a global scale, among other things. As described in the document, simulating all these phenomena is a very complex task and many simplifications have been made in order to make this project viable.

The ultimate goal is not to try to simulate the planet Earth, as it is a great complex task and with countless variables, instead, we want to explore the idea of implementing in a simplified way what we know about our planet, and others, to obtain approximate results that can be extrapolated to see what other habitable planets like ours might look like, or create totally unique others with very different characteristics.

Keywords (10 maximum):

Graphic simulator	Shaders	Computer graphics	Terrestrial planets
Planetary evolution	Plate tectonics	Hydraulic erosion	Atmosphere
OpenGL	Climate		

SUMARI

ANNEXOS

1. Annex 1: Fitxer amb el codi del programa
   TFG Annexos - Implementació d'un generador procedural de planetes (Oriol Fernandez).docx

SUMARI DE FIGURES

GLOSSARI DE SIGNES, SÍMBOLS, ABREVIATURES, ACRÒNIMS I TERMES

FBM	*Fractional Brownian Motion*
MSLP	Pressió mitjana a nivell del mar
ITCZ	Zona de convergència intertropical
STHZ	Zona d'alta pressió subtropical
PF	Front polar

INTRODUCCIÓ

És difícil fer-se una idea de la quantitat de temps que va ser necessari perquè la Terra es converteixi en el que coneixem avui dia (milers de milions d'anys). En comparació, es podria dir que la història de la humanitat just acaba de començar.

Com que som incapaços d'assimilar aquestes enormes escales de temps, podem comprimir tot aquest període des que es va formar el sistema solar fins a l'actualitat en una simulació d'un o dos minuts, suficient per veure tots aquests fenòmens que han intervingut en el modelat del nostre planeta en funcionament. Això és exactament el que és capaç de fer aquest programa.

Perquè la simulació sigui fluida s'han de realitzar càlculs molt complexos i costosos 30, 60 o més vegades cada segon. Aquí és on aprofitem la capacitat computacional de les targetes gràfiques que incorporen els dispositius moderns. Aquest component té la funció principal de processar les dades que rep del processador i transformar-les en informació que es representa de forma visual en monitors. Els càlculs que fem es poden distribuir en els milers de nuclis que tenen aquestes targetes, ja que estan dissenyades per aquest tipus de càrrega de treball, a diferència de les CPU.

El programa ofereix la possibilitat d'interaccionar i mostrar de forma molt visual el que es coneix respecte a l'evolució dels planetes i els processos que intervenen. En aquests només veurem la punta de l'iceberg, pel fet que molts són conceptes complexos amb molta informació, però s'ha tractat de focalitzar el contingut al màxim per obtenir els millors resultats amb una gran simplificació en el temps disponible.

Aquesta idea va sorgir del treball en el qual ha treballat en els últims anys David A. Roberts[7], desenvolupador de software i artista digital. Que te una bona experiència en el mon del *shaders* i ha anat desenvolupant per separat simulacions físiques de tot tipus i també d'aquests fenòmens que finalment va unificar en una única simulació de la Terra molt visual.

A continuació es presenten els capítols on es parla dels esdeveniments a través del temps que van portar a cada un dels aspectes que volem simular. Per a cada un, es dona una introducció teòrica on s'expliquen les bases que es volen simular per després passar a la part pràctica on s'implementen els conceptes en forma de codi.

El primer capítol servirà per a conèixer l'entorn en el qual es treballa i com està estructurat el projecte. Al segon capítol s'introdueix el primer gran element de la simulació, s'explica la primera etapa de formació del planeta i en acabar tenim una base la qual anem ampliant amb més característiques. En el tercer capítol, es presenta la formació dels oceans, com va arribar l'aigua a la superfície terrestre i també veurem per primer cop un element clau a la simulació, el pas del temps i com representar-lo en el programa. Lligat amb l'anterior, en el capítol 4 veurem els efectes que ocasiona l'aigua al planeta i com transforma el seu paisatge amb rius i l'erosió de l'aigua. Al capítol 5 es presenta l'origen i funcionament de les plaques tectòniques al planeta i com simular la interacció entre elles de forma realista. Continuant al capítol número 6, es presenta la formació de l'atmosfera i veurem el seu funcionament en detall per entendre els fenòmens que causen que el cel tingui el seu color característic i veurem perquè es torna vermell quan el sol està baix en l'horitzó, i com podem implementar tots aquests càlculs necessaris tan costosos de forma eficient entre altres coses. Per acabar veurem el funcionament del clima a escala global en el capítol 7, explorarem aspectes com la pressió atmosfèrica, els corrents del vent, temperatura, precipitacions i veurem com implementar aquests fenòmens de forma que puguem obtenir resultats prou realistes.

Finalment, al capítol 8 es presenten altres elements importants del programa, i per acabar, a l'últim capítol veurem quines millores es podrien implementar per a poder ampliar el treball.

OBJECTIUS DEL TFE

L'objectiu d'aquest treball és principalment poder visualitzar l'evolució que va experimentar el nostre planeta fins a arribar a l'actualitat, amb la informació que es coneix actualment.

Tractarem de simular els processos més significatius, però només com a una primera aproximació, ja que, no només existeixen una gran quantitat d'aquest, sinó que són realment complex si s'estudien en profunditat. D'igual manera, si volguéssim ser molt precisos en tots aquests càlculs, la simulació d'alguns elements seria massa costosa computacionalment i no podríem veure el resultat en temps real.

A part, és interessant explorar el potencial que ens dona desenvolupar un model d'aquestes característiques introduint la possibilitat d'interactuar i modificar els paràmetres que defineixen els aspectes de cada simulació. De forma que es pugui veure com afecten aquestes modificacions a l'aspecte final. Algunes d'aquestes opcions podrien ser la mida i velocitat de les plaques tectòniques, poder modificar l'altura del terreny, temperatura, atmosfera entre altres.

Això es pot ampliar cap a múltiples direccions com per exemple generar diferents tipus de satèl·lits, exoplanetes i similars.

Des d'un principi tenia clar que per a fer funcionar aquesta simulació, l'ús d'ombrejadors (*shaders*) era una opció molt potent i flexible. Però un dels principals inconvenients era fer servir OpenGL, ja que és una tecnologia que es va llançar per primer cop l'any 1992 i volia poder executar el codi en qualsevol equip fàcilment.

CAPÍTOL 1. INTRODUCCIÓ A L'ENTORN 3D

En aquest capítol es presenta una introducció a les tecnologies que formen l'entorn on es generen els gràfics 3D per poder visualitzar i interactuar amb les simulacions. També veurem com preparar pas a pas un entorn bàsic per començar a desenvolupar l'aplicació i com és l'estructura i funcionament d'alguns dels components principals.

1.1 THREE.JS

Three.js és una biblioteca JavaScript llançada l'any 2010 que s'utilitza per crear i mostrar gràfics 3D en un navegador web mitjançant WebGL una altra API de gràfics a baix nivell creada específicament per a web, que alhora deriva de OpenGL[10].. La renderització s'executa a la unitat de processament gràfic (GPU), el que significa que el rendiment en temps d'execució és molt bo. [1].

WebGL és compatible amb les versions més recents d'Internet Explorer, Chrome, Firefox i Safari. Malauradament, no tots els navegadors mòbils són compatibles. Per descomptat, el rendiment en dispositius mòbils és substancialment inferior al d'escriptori. Però tècnicament, WebGL és en realitat OpenGL ES, que és un subconjunt d'OpenGL destinat a dispositius mòbils.

La biblioteca Three.js és un fitxer independent de JavaScript. Pot ser inclòs al nostre projecte a través d'un servidor CDN (Content Delivery Network) o directament d'una còpia local.

Una aplicació three.js requereix que es creïn un munt d'objectes i es connectin. Aquí teniu un diagrama que representa una petita aplicació three.js

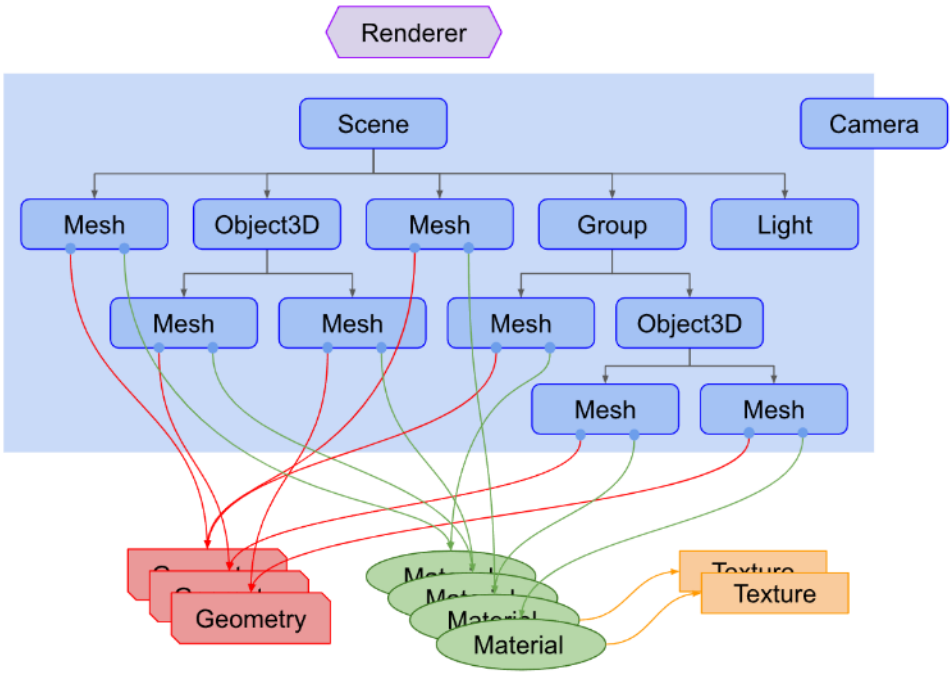

Figura 1 Diagrama de l'estructura de Three.JS. Font: [10]

El *Renderer* és possiblement l'objecte principal de three.js. Pots passar una escena i una càmera a un renderitzador i aquest representa (dibuixa) la part de l'escena 3D que es troba dins del tronc de la càmera com a imatge 2D a un *canvas* (llenç).

Hi ha una escenografia, que és una estructura en forma d'arbre, que consta de diversos objectes com un objecte d'escena, diversos objectes *mesh* (malla), objectes de llum, grup, *Object3D* i objectes de càmera.

Un objecte Escena defineix l'arrel de l'escenografia i conté propietats com el color de fons i la boira. Aquests objectes defineixen una estructura jeràrquica com a arbre pare/fill i representen on apareixen els objectes i com s'orienten. Els fills estan posicionats i orientats en relació amb els seus progenitors.

Els objectes *mesh* (malla) representen dibuixar una geometria específica amb un material específic.

Tant els objectes de material com els de geometria poden ser utilitzats per diversos objectes de malla. Per exemple, per dibuixar dos cubs blaus en diferents ubicacions necessitem dos objectes *Mesh* per representar la posició i l'orientació de cada cub. Només necessitaríem una geometria per contenir les dades del vèrtex d'un cub i només necessitaríem un material per especificar el color blau. Tots dos objectes de malla poden fer referència al mateix objecte de geometria i al mateix objecte de material.

Els objectes de geometria representen les dades del vèrtex d'alguna peça de geometria com una esfera, un cub, un pla, un gos, un gat, un humà, un arbre, un edifici, etc... Three.js proporciona molts tipus de geometria primitius. També podeu crear geometria personalitzada així com carregar geometria dels fitxers.

Els materials representen les propietats de la superfície que s'utilitzen per dibuixar la geometria, incloses coses com el color a utilitzar i la seva brillantor. Un material també pot fer referència a un o més objectes de textura que es poden emprar, per exemple, per dibuixar una imatge a la superfície d'una geometria[10].

Podem trobar alguns exemples d'ús de la llibreria a la indústria, directament des de la pàgina web de la llibreria. El primer (Figura 2) és el cas de GitHub, el famós servei de allotjament de repositoris Git i control de revisió, que ofereix una pantalla principal interactiva desenvolupada amb aquesta tecnologia.

Figura 2 Pàgina web de Github. Font: [41]

En aquest segon cas, a la Figura 3 veiem una simulació en temps real de l'entrada, descens i aterratge de la missió *Mars 2020* de la *NASA*.

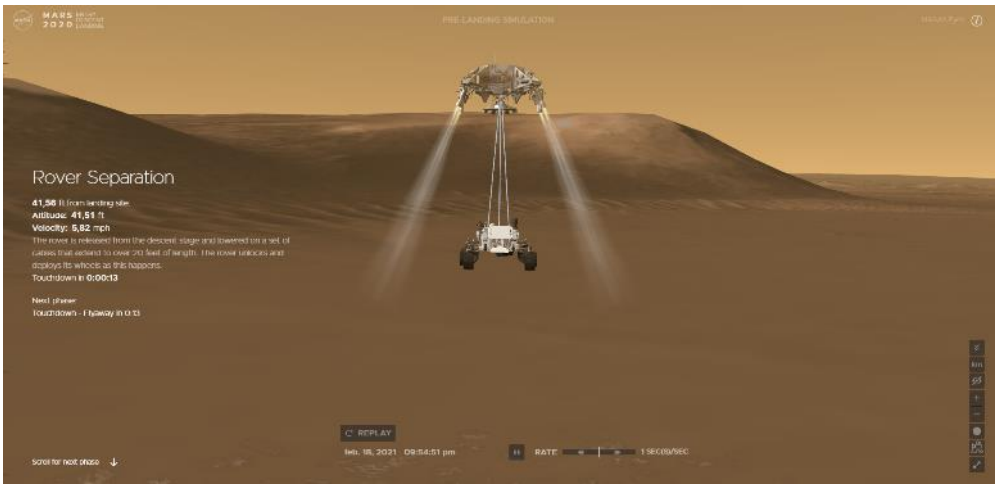

Figura 3 Simulació Mars 2020 de la NASA. Font: [42]

Per acabar, a la Figura 4 es mostra un simulador per familiaritzar-te amb els controls de la interfície real que fa servir els astronautes de la *NASA* per a pilotar manualment la càpsula *Dragon 2* de *SpaceX* fins a l'estació espacial internacional.

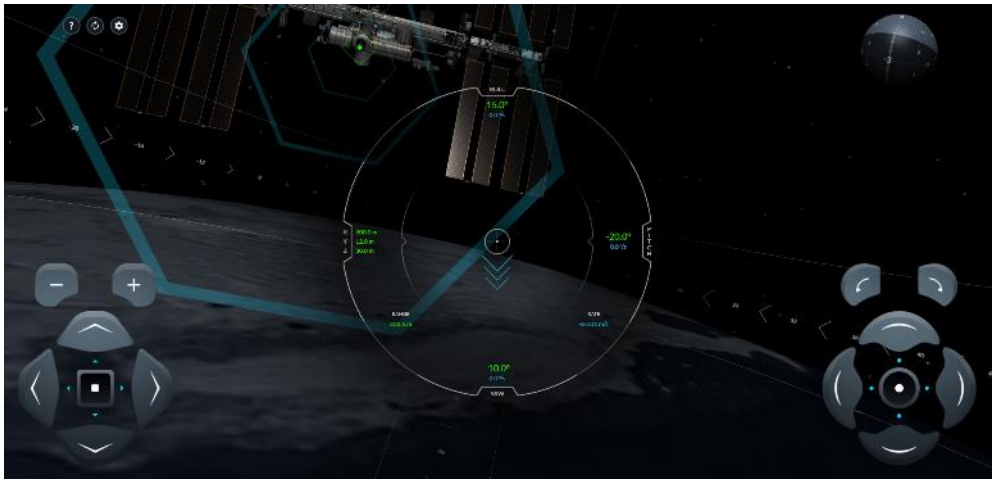

Figura 4 Simulador Dragon 2 de SpaceX. Font: [43]

De les opcions i tecnologies disponibles per desenvolupar el projecte he escollit Three.js, ja que després d'haver treballat amb OpenGL pur amb C++, volia provar un flux de treball diferent, pràctic, i amb el que sigui fàcil d'interactuar. Llavors vaig descobrir WebGL, que deriva de OpenGL per poder portar aquests gràfics als navegadors.

A més, Three.js és una de les llibreries més famoses que hi ha, és fàcil d'aprendre, té una gran comunitat, bona documentació i bon rendiment.

1.2 SHADERS

Els *shaders* són un conjunt d'instruccions que s'executen alhora per a cada píxel de la pantalla. Això vol dir que el codi que s'escriu s'ha de comportar de manera diferent segons la posició del píxel a la pantalla. El programa funciona com una funció que rep una posició i retorna un color, i quan es compila, s'executarà extraordinàriament ràpid.

Els *shaders* funcionen ràpidament gràcies a la paral·lelització de processos. Si imaginem la CPU de l'ordinador com una gran canonada i cada tasca com un objecte que passa a través, com una línia de fàbrica. Algunes tasques són més grans que altres, la qual cosa significa que requereixen més temps i energia per afrontar-les, és a dir, requereixen més potència de processament. A causa de l'arquitectura dels ordinadors, els treballs es veuen obligats a executar-se en sèrie; cada treball s'ha d'acabar d'un en un. Els ordinadors moderns solen tenir grups de quatre, sis, vuit o més processadors que funcionen com aquestes canonades, completant les tasques una darrere l'altra per mantenir les coses sense problemes. Cada tub també es coneix com a fil[1].

Cada píxel de la pantalla representa una petita tasca senzilla. Individualment, cada tasca de píxel no és un problema per a la CPU, però (i aquí és el problema) la petita tasca s'ha de fer a cada píxel de la pantalla. Això vol dir que en una pantalla antiga de 800 x 600, s'han de processar 480.000 píxels per fotograma, la qual cosa significa 14.400.000 càlculs per segon. Aquest és un problema prou gran per a sobrecarregar un microprocessador. En una pantalla retina moderna de 2880 x 1800 que funciona a 60 fotogrames per segon, aquest càlcul suma 311.040.000 càlculs per segon.

Aquí és on entra el processament paral·lel. En lloc de tenir un parell de microprocessadors o canonades grans i potents, és més intel·ligent tenir molts microprocessadors petits que funcionin en paral·lel al mateix temps. Això és el que és una unitat de processador gràfic (GPU).

Un altre ús de la GPU són les funcions matemàtiques especials accelerades mitjançant maquinari, de manera que les operacions matemàtiques complicades es resolen directament amb els microxips en comptes d'amb el programari. Això significa que les operacions trigonomètriques i matricials seran molt ràpides.

Els *shaders* s'escriuen en el llenguatge GLSL (OpenGL Shading Language), que és l'estàndard específic dels programes *shader* i és semblant a C. Hi ha dos tipus de *shaders*: Vèrtex *shader*, que calcula la posició i els atributs de cada vèrtex del polígon, i fragment shaders, que calcula el color de cada píxel.

1.3 ESCENA INICIAL

Ara que hem vist com funcionen les tecnologies informàtiques que farem servir, anem a veure com es pot començar l'entorn de treball sobre el qual treballarem i anirem ampliant en cada apartat.

Partim d'una escena simple amb una càmera, un *renderer*, que s'encarrega de renderitzar els elements en pantalla amb WebGL i una geometria esfèrica senzilla que ens servirà per simular el nostre planeta.

Per visualitzar aquesta escena necessitem una pàgina web on renderitzar-la. Per tant, cal un fitxer HTML que contingui, entre altres coses, un *div* que serà on es carregui el *canvas*, que és el que veurem per pantalla. També fem la crida al script inicial app.js i declarem un fitxer d'estils CSS.

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Procedural Planet Generator</title>
    <link rel="stylesheet" href="src/css/style.css">
  </head>

  <body>
    <div id="container"></div>
    <script type="module" src="app.js"></script>
  </body>

</html>
```

El fitxer app.js el comencem important la llibreria de THREE

```
import * as THREE from 'https://cdn.skypack.dev/three@0.136';
```

Per qüestions d'organització, farem ús de classes, de forma que un cop declarada, cal fer la crida per executar-la. L'estructura que tenim és un constructor i una funció render que es crida contínuament per actualitzar el *canvas*.

```
export default class app {
  constructor () {...}
  render() {...}
}

new app();
```

Al constructor declarem el *renderer* de WebGL, configurem les mides segons la finestra on s'està visualitzant i l'afegim al contenidor HTML que hem creat prèviament perquè injecti el codi necessari per visualitzar al navegador l'escena.

També crearem una càmera que utilitza la projecció en perspectiva, aquesta està posicionada en l'escena i seran els nostres ulls. Aquí podem configurar alguns paràmetres com el camp de visió de la càmera.

```
constructor () {
  this.renderer = new THREE.WebGLRenderer();
  this.renderer.setSize( window.innerWidth, window.innerHeight );
  document.getElementById('container').appendChild(
this.renderer.domElement );

  this.camera = new THREE.PerspectiveCamera( 70, window.innerWidth /
window.innerHeight, 0.01, 10 );
  this.camera.position.z = 2;
  this.scene = new THREE.Scene();
  this.addMesh();
  this.render();
}
```

El primer paràmetre indica el *FOV*. *FOV* és l'abreviatura de *field of view* (camp de visió). En aquest cas de 70 graus en la dimensió vertical. Tingueu en compte que la majoria dels angles de three.js estan en radians, però per alguna raó la càmera de perspectiva pren graus.

El segon paràmetre és la relació d'aspecte de visualització del llenç. En aquest cas farem que ocupi tota la finestra.

Segueixen els paràmetres *near* (prop) i *far* (lluny). Representen l'espai davant de la càmera que es renderitzarà. Qualsevol cosa abans d'aquest interval o després d'aquest interval es retallarà (no es dibuixarà).

Aquests quatre paràmetres defineixen el tronc o *frustum*. Un tronc és el nom d'una forma 3D que és com una piràmide amb la punta tallada, com veiem a la Figura 5

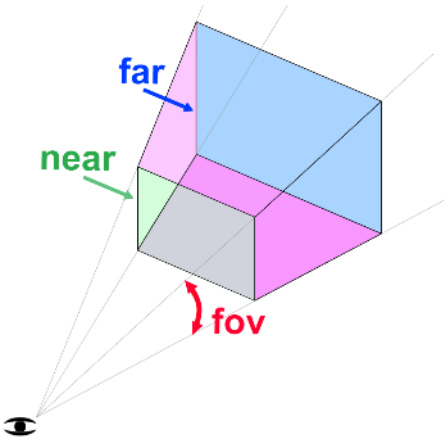

Figura 5 Tronc geomètric que representa l'espai de visió de la cámara. Font: [10]

Al constructor també fem la crida a una funció on crearem la geometria i material que conjuntament formen una malla (*mesh*) i s'afegeix a l'escena per renderitzar-la. Three.js ens

ofereix molts tipus de geometries per escollir, en aquest cas fem ús de *SphereGeometry*, que com el nom indica, és una esfera. El seu constructor ens deixa definir alguns paràmetres com la mida i el nombre de segments horitzontals i verticals. Com més gran sigui aquest valor, més vèrtex tindrà la geometria.

Els materials descriuen l'aspecte dels objectes, i també tenim moltes opcions per escollir (*NormalMaterial*, *PhongMaterial*, *StandardMaterial*, *ShaderMaterial*, etc...). Per començar, fem servir un material que mapeja els vectors normals amb colors RGB per identificar la geometria i també perquè no necessita il·luminació per poder veure el resultat a l'escena, un *MeshNormalMaterial*.

Finalment, es crea un *mesh* a partir de la geometria i el material i s'afegeix a l'escena.

```
addMesh() {
  this.geometry = new THREE.SphereBufferGeometry( 1, 32, 32 );
  this.material = new THREE.MeshNormalMaterial();
  this.mesh = new THREE.Mesh( this.geometry, this.material );
  this.scene.add( this.mesh );
}
```

Render és la funció que s'encarrega de crear un bucle que farà que el renderitzador dibuixi l'escena cada vegada que s'actualitza la pantalla (En una pantalla típica això significa 60 vegades per segon). Amb el requestAnimationFrame el que aconseguim és que aquest bucle s'aturi quan l'usuari navega a una altra pestanya del navegador, per tant, no malgasta la potència de processament ni bateria.

```
render() {
  this.renderer.render( this.scene, this.camera );
  window.requestAnimationFrame(this.render.bind(this))
}
```

El resultat és el que veiem a la Figura 6, una escena simple amb res més que una esfera de colors. A partir d'aquí, anirem ampliant i modificant el codi introduint diferents elements.

Figura 6 Escena base amb una esfera. Font: pròpia

CAPÍTOL 2. PERÍODE DE FORMACIÓ INICIAL

En aquest capítol es presenta la primera fase de formació del planeta, primerament en una explicació basada en la teoria en la qual ens basem per posteriorment tractar de replicar de forma aproximada els esdeveniments que va experimentar la Terra en la seva evolució temporal i plasmar-lo en el nostre entorn.

2.1 FORMACIÓ DE LA TERRA

Els cossos del sistema solar es van formar fa més de 4.600 milions d'anys a partir d'una barreja de gasos, principalment hidrogen, barrejat amb petites quantitats d'elements més pesats formats a les estrelles més antigues, i una mica de pols addicional.

La contracció gravitatòria d'aquesta massa i la formació del disc protoplanetari a causa de la conservació del moment angular van ser els primers passos en la formació del sistema. Mentre que la major part de la massa es concentrava al centre de gravetat del sistema, formant el Sol, aproximadament un 1% va romandre al disc, convertint-se en la matèria primera per formar els planetes, satèl·lits i altres cossos menors. [14]

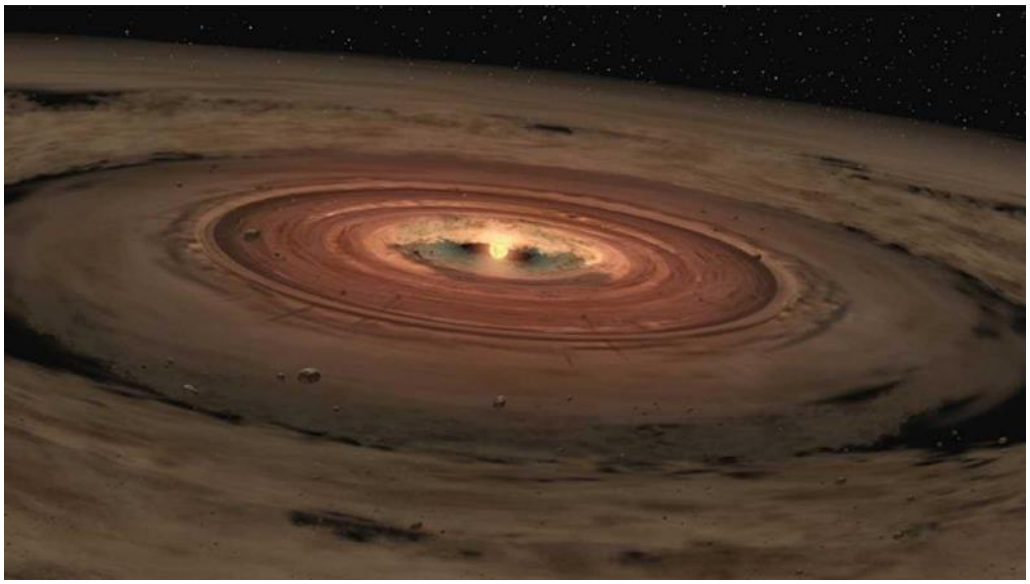

Figura 7 Representació artística d'un disc protoplanetari. Font: [11]

La temperatura en d'aquest disc protoplanetari no era uniforme. Com que diferents materials es condensen a diferents temperatures, el nostre sistema solar va formar diferents tipus de planetes. La línia divisòria dels diferents planetes del nostre sistema solar s'anomena línia de congelació o *frost line*.

Els compostos d'hidrogen, com l'aigua i el metà, solen condensar-se a baixes temperatures i romanen gasosos dins de la línia de congelació on les temperatures són més altes. Els materials rocosos i metàl·lics més pesats són més adequats per condensar-se a temperatures més altes. Així, els planetes interiors estan fets gairebé íntegrament de roca i metall i formen el grup conegut com els planetes terrestres.

Els elements i minerals més pesats es van condensar en trossos sòlids de roca, tots orbitant al voltant del Sol a la mateixa velocitat. Com podem imaginar, les col·lisions d'objectes que es mouen a la mateixa velocitat són menys destructives que les d'objectes que es mouen a diferents velocitats. Així, quan les roques que orbiten el Sol es mouen a prop les unes de les altres, s'enganxen més sovint que no pas es destrueixen. Aquestes peces creixen gradualment en un procés anomenat acreció. Un cop són prou grans, la gravetat els obliga a adoptar formes esfèriques. [17]

Fora de la línia de congelació, les temperatures són més baixes i els compostos d'hidrogen poden condensar-se en gels. La roca i el metall encara estan presents al sistema solar exterior, però tots dos estan superats en nombre i superats pels compostos d'hidrogen. Així, els planetesimals que es van formar al sistema solar exterior estan compostos principalment per compostos d'hidrogen amb traces de roca i metall. L'hidrogen i l'heli no es condensen a la nebulosa solar, i són bastant abundants a les grans òrbites dels objectes del sistema solar exterior. A mesura que els planetesimals exteriors van continuar creixent, la força de la seva gravetat es va fer més forta. El material que l'envolta, principalment l'hidrogen i l'heli, és cada cop més atret pels planetesimals a mesura que creixen de mida i els planetesimals s'acumulen cada cop més.

Els planetesimals jovians aviat es van convertir en els nuclis gelats i densos que veiem avui envoltats d'enormes núvols de gas acumulat. Igual que el col·lapse de la nebulosa solar, aquestes boles de gas poden créixer prou grans com per induir el col·lapse gravitatori. Això implica escalfar-se, aplanar-se i girar més ràpidament. És possible que a mesura que els protoplanetes jovians es van col·lapsar, partícules més petites del disc circumdant es van formar en algunes de les llunes que ara orbiten els planetes exteriors individuals. Això té sentit, ja que tots els planetes exteriors tenen moltes llunes i anells que orbiten en el mateix pla, igual que els planetes del nostre sistema solar orbiten al voltant del Sol en el mateix pla. [17]

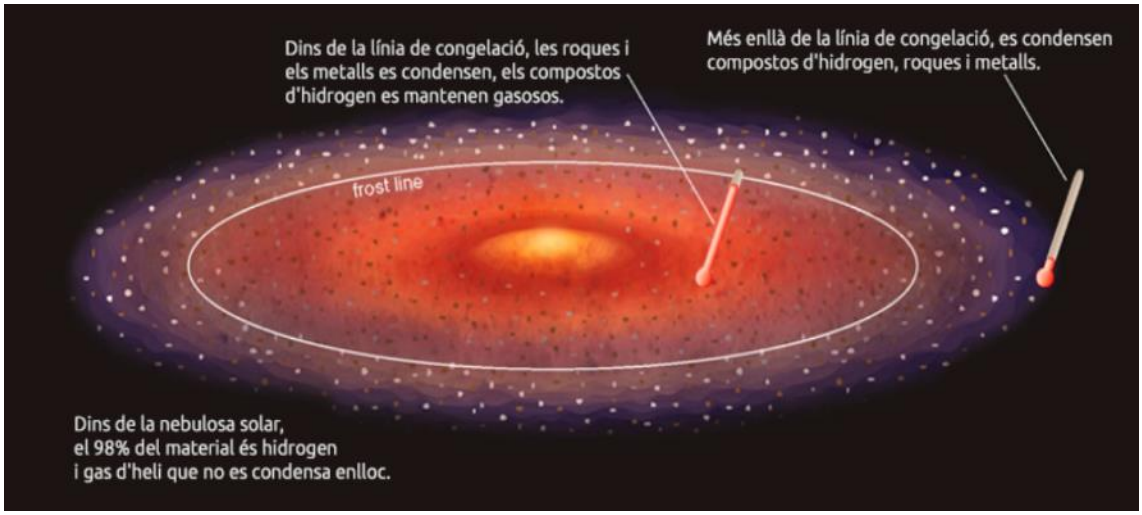

Figura 8 Diagrama de la línia de congelació. Font: [17]

Els planetes terrestres com la Terra es va anar formant gràcies a aquestes col·lisions entre partícules de pols, asteroides i altres planetes en creixement, inclòs un darrer impacte gegant que va llançar prou roca, gas i pols a l'espai per formar la lluna.

Encara que les roques que registren les primeres parts de la història de la Terra han estat destruïdes o deformades per més de quatre mil milions d'anys de geologia, els científics poden utilitzar roques modernes, mostres de la lluna i meteorits per esbrinar quan i com es va formar la Terra i la lluna, i quina era la seva aparença llavors. [11]

Els meteorits han sigut clau per arribar a aquesta conclusió, ja que porten molts tipus de material de tot el sistema solar a la Terra, on els científics poden estudiar-los. Aquests materials inclouen còndrules, trossos de pols i roca que han sobreviscut abans que els planetes es formessin, i trossos d'asteroides i planetesimals deixats enrere pel procés de construcció de planetes.

Els elements radioactius com l'urani i l'hafni estan atrapats dins dels minerals que componen aquests objectes quan es formen, la qual cosa permet als científics planetaris saber com d'antics són.

Utilitzant aquestes mesures i simulacions de la física de la pols i les col·lisions planetesimals, els científics planetaris i els astrònoms han establert que el procés de pols a protoplaneta dura desenes de milions d'anys.

La Terra primerenca estava coberta per un oceà de magma: una capa de roca fosa de centenars de quilòmetres de profunditat. Qualsevol aigua present només existiria com a vapor d'aigua a l'atmosfera.[11]

Figura 9 Representació artística de la Terra en el periode Hadeà. Font: [3]

Els elements pesats van començar a enfonsar-se més enllà dels oceans i el magma cap al centre del planeta. A mesura que això ocorria, la Terra es va diferenciar en capes, amb la capa més externa com una coberta sòlida de material relativament més lleuger, mentre que el material fos més dens es va enfonsar al centre.[2]

Investigacions mostren que durant aquest període, la superfície de la Terra va ser fortament reprocessada (o fosa, barrejada i enterrada) com a resultat dels impactes d'asteroides gegants.[4]

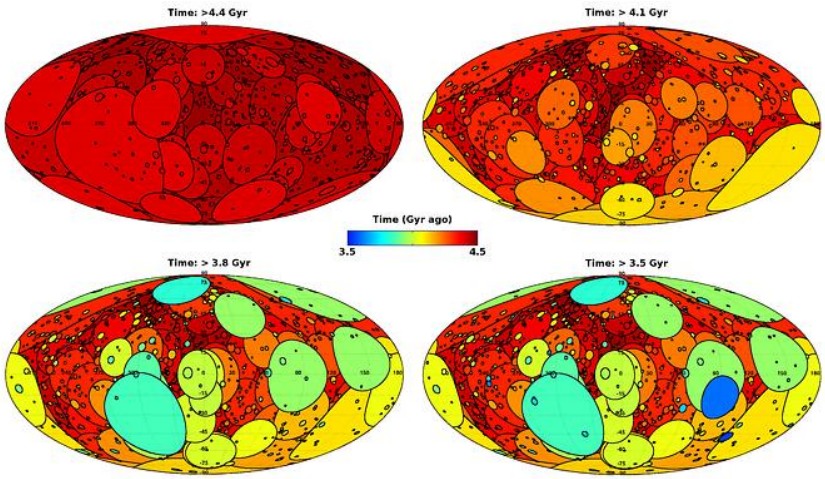

Figura 10 Distribució espacial i mides dels cràters formats a la Terra primitiva. Cada cercle indica la mida final estimada del cràter. La codificació de colors indica el moment de l'impacte. Font: [4]

Aquestes col·lisions no només van alterar severament la geologia de la Terra, sinó que probablement també van tenir un paper important en l'evolució posterior de la vida a la Terra.[4]

Quan un asteroide impacta contra una superfície planetària, viatja normalment a diverses desenes de quilòmetres per segon, moltes vegades la velocitat del so. Tot i que la depressió resultant pot tenir una certa semblança amb el forat que resulta de llançar una pedra a una caixa de sorra, el procés físic que es produeix és en realitat molt més proper al de l'explosió d'una bomba atòmica. Un gran impacte de meteorit allibera una quantitat enorme d'energia cinètica en una petita àrea en poc temps.[5]

Immediatament després d'impactar a la superfície del planeta, les ones de xoc s'imparteixen tant al material de la superfície com al propi meteorit. A mesura que les ones de xoc s'expandeixen al planeta i al meteorit, dissipen energia i formen zones de material vaporitzat, fos i triturat cap a l'exterior des d'un punt per sota de la superfície del planeta que és aproximadament tan profund com el diàmetre del meteorit. El meteorit normalment es vaporitza completament per l'energia alliberada.[5]

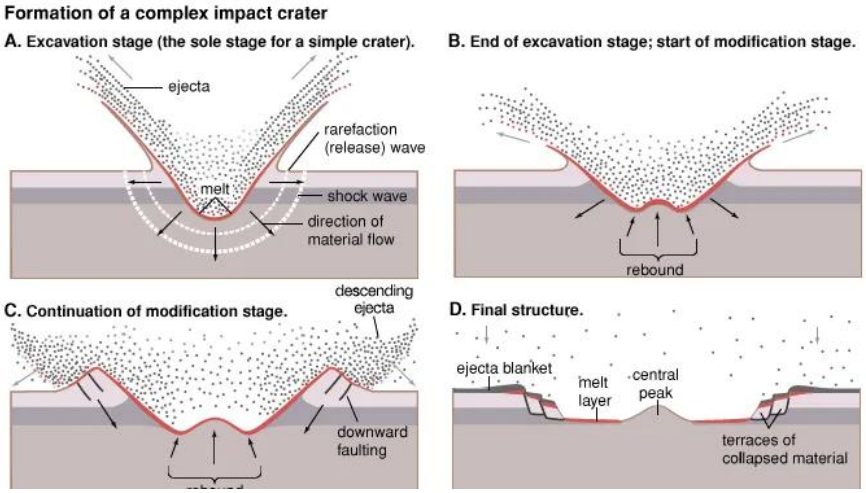

Figura 11 **Formació d'un cràter d'impacte complex. Font: [5]**

Quan el cràter és relativament petit, la seva formació s'acaba quan s'atura l'excavació. La forma de relleu resultant s'anomena cràter simple. Els cràters més petits no necessiten més d'uns segons per formar-se completament, mentre que els cràters que tenen desenes de quilòmetres d'ample probablement es formen en pocs minuts.[5]

A mesura que els cràters de meteorits es fan més grans, el procés de formació no s'atura amb l'excavació. Per a aquests cràters, el forat parabòlic és aparentment massa gran per suportar-se i col·lapsa en un procés que genera una varietat de característiques. Aquest procés de col·lapse s'anomena etapa de modificació, i la depressió final es coneix com a cràter complex. [5]

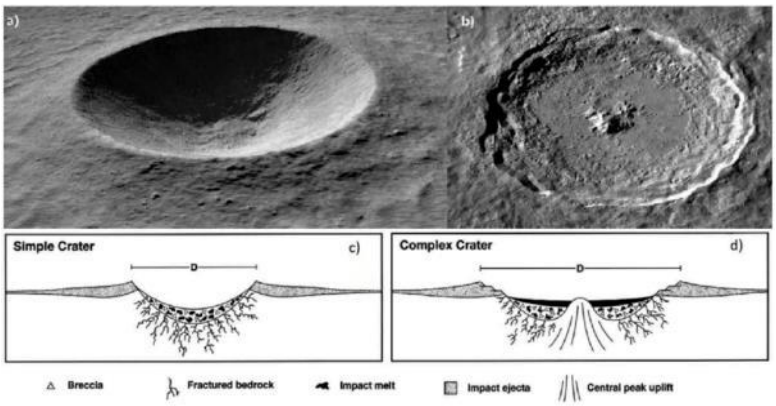

Figura 12 **Exemple de diferents tipus de cràters lunars. Font: [6]**

2.2 IMPLEMENTACIÓ

Per a replicar com era el planeta al seu període inicial, la primera tasca és generar un mapa d'aquest terreny, això ho farem de forma procedural, sense cap textura pre-generada.

Per a aquesta tasca farem ús de *shaders*, que ens donen molta flexibilitat. Per tant, el primer que cal fer és crear i definir els fitxers.

2.2.1 VERTEX SHADER

Recordem que la seva funció és aplicar el codi a cada vèrtex d'una malla de la nostra geometria. Normalment s'utilitza per distorsionar o animar la seva forma. Dins del nostre fitxer tindrem:

```
const vertex = /* glsl */ `
  varying vec2 vUv;
  varying vec3 vPosition;

  void main() {
    vUv = uv;
    vPosition = position;
    gl_Position = projectionMatrix * modelViewMatrix * vec4( position, 1.0 );
  }
`;
export default vertex;
```

El primer que cal observar és que tot el nostre codi GLSL està en una cadena. Ho fem perquè WebGL passarà aquest fragment de codi a la nostra GPU i hem de passar el codi a WebGL dins de Javascript. La segona cosa que es pot notar és que estem utilitzant variables que no hem creat. Això es deu al fet que Three.js ens passa aquestes variables a la GPU de forma predeterminada.

Dins d'aquest fragment de codi calculem on s'han de col·locar els vèrtexs de la nostra malla. Ho fem calculant on es troben els vèrtex de l'escena multiplicant la posició de la malla a l'escena (modelViewMatrix) i la posició del vèrtexs. Després d'això, multipliquem aquest valor amb la posició de la càmera relativa amb l'escena (*projectionMatrix*) de manera que el nostre *shader* respecta la configuració de la càmera dins de Three.js. La variable *gl_Position* és el valor final que pren la GPU per dibuixar els nostres vèrtex.

2.2.2 FRAGMENT SHADER

Aquest codi s'aplica a cada fragment de la nostra malla. Un fragment és el resultat d'un procés anomenat rasterització, que converteix tota la malla en una col·lecció de triangles. Per cada píxel que està cobert per la nostra malla hi haurà almenys un fragment. El *fragment shader* s'utilitza normalment per fer transformacions de color en píxels.

El que farem serà dibuixar sobre la nostra geometria amb aquest shader. Els mateixos cràters es generen mitjançant una variació d'un tipus de soroll anomenat soroll cel·lular (o worley). Aquest es basa en els diagrames de Voronoi [1], que per definició es tracta de dividir l'espai en regions cel·lulars on tots els punts de cada regió estan més a prop del seu punt definitiu que qualsevol altre punt [12]. Cada polígon conté exactament un punt generador i cada punt d'un polígon donat està més a prop del seu punt de generació que de qualsevol altre. Un diagrama de Voronoi de vegades també es coneix com a tessel·lació de Dirichlet. Les cèl·lules s'anomenen regions de Dirichlet, polítops de Thiessen o polígons de Voronoi. [13]

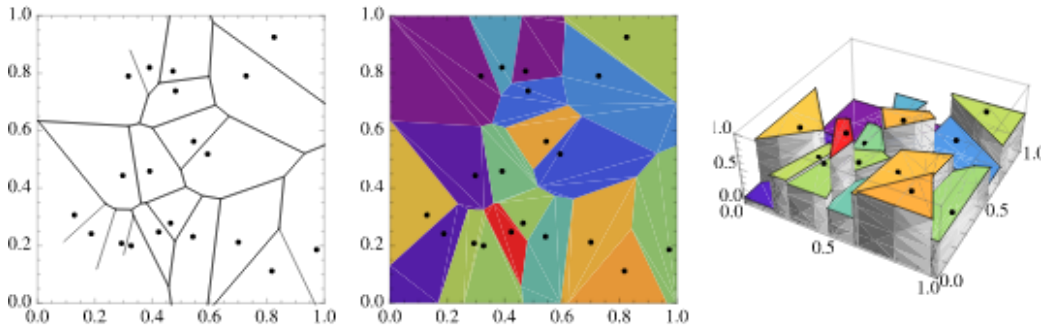

Figura 13 Diagrames de Voronoi. Font: [13]

Aquest algorisme també es pot interpretar des de la perspectiva dels punts i no dels píxels. En aquest cas es pot descriure com: cada punt creix fins que troba l'àrea de creixement des d'un altre punt. Això reflecteix algunes de les regles de creixement de la natura. Les formes vives estan modelades per aquesta tensió entre una força interior per expandir-se i créixer, i les limitacions de forces externes. L'algorisme clàssic que simula aquest comportament rep el nom del matemàtic ucraïnès Gueorgui Voronói (1868-1908)[1].

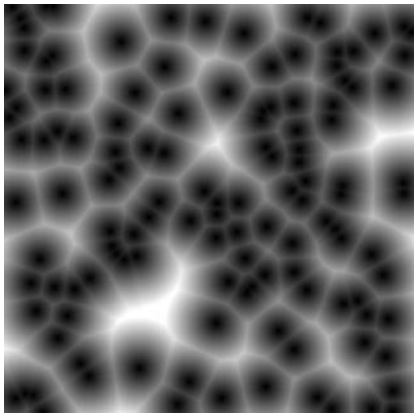

Figura 14 Exemple de soroll cel·lular. Font:[1]

El soroll cel·lular es basa en camps de distància, la distància al més proper d'un conjunt de punts característics. Suposem que volem fer un camp de distància de 4 punts. Què hem de fer? Bé, per a cada píxel volem calcular la distància al punt més proper. Això vol dir que hem de recórrer tots els punts, calcular les seves distàncies al píxel actual i emmagatzemar el valor del qual està més a prop?[1]. No, en comptes d'iterar cada un dels punts, podem simplement iterar el més proper si dividim l'espai en cel·les. No totes, només les que estan al voltant de l'actual.

Això vol dir de -1 (esquerra) a 1 (dreta) a l'eix x i -1 (a baix) a 1 (a dalt) a l'eix y. Es pot iterar una regió de 3x3 de 9 cel·les utilitzant un bucle for doble com aquest:

```
for (int y= -1; y <= 1; y++) {
    for (int x= -1; x <= 1; x++) {
        // Lloc veí a la quadrícula
        vec2 neighbor = vec2(float(x),float(y));
        ...
    }
}
```

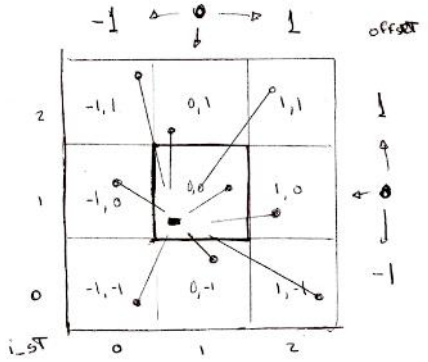

Figura 15 Espai dividit per iterar cel·les veïnes. Font: [1]

Amb aquesta idea com a base, creem una funció que ens permeti obtenir una distribució de cràters realista. Per evitar la regularitat visible, els centres del cràter reben un desplaçament pseudoaleatori dels punts de la quadrícula, utilitzant una funció *hash*. Per calcular la influència d'un cràter en un lloc donat, es fa una mitjana ponderada dels cràters que pertanyen als punts de la quadrícula propers, amb pesos que disminueixen exponencialment amb la distància des del centre. Les vores del cràter es generen per una simple corba sinusoidal.

```
float craters(vec3 x) {
  vec3 p = floor(x);
  vec3 f = fract(x); // Same as x - floor(x)
  float va = 0.;
  float wt = 0.;
  for (int i = -2; i <= 2; i++) {
    for (int j = -2; j <= 2; j++) {
      for (int k = -2; k <= 2; k++) {
        vec3 g = vec3(i,j,k);
        vec3 o = 0.8 * hash33(p + g);
        float d = distance(f - g, o);
        float w = exp(-4. * d);
        va += w * sin(2.*PI * sqrt(d));
        wt += w;
      }
        }
    }
    return abs(va / wt);
}
```

El resultat d'aplicar aquesta funció és el següent:

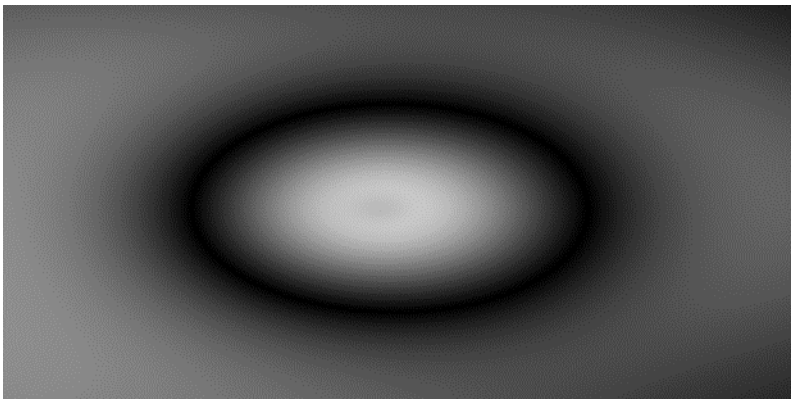

Figura 16 Resultat del *fragmentShader* amb la funció crater(). Font: pròpia

Com hem vist prèviament, els asteroides tenen una varietat de mides, també la tenen els cràters resultants. Per adaptar-se a això, el *shader* itera sobre múltiples nivells de detall, superposant cràters de mida decreixent els uns sobre els altres.

El resultat és l'exemple següent:

```
// Creates a heightmap of a terrain with craters
float cratersHeightMap(vec3 p) {
    float height = 0.;

    // Multiple passes various crater sizes
    for (float i = 0.; i < 5.; i++) {

      // Generate the craters
      float c = craters(vec3(0.5 * pow(2.2, i) * p));

      // Generate the FBM noise
      float noise = 0.4 * exp(-3. * c) * FBM( vec3(10. * p) );

      // Constrain a value to lie between two further values
      float x = 3. * pow(0.4, i);
      float min = 0.;
      float max = 1.;
      float w = clamp(x, min, max);

      // Mix and add the result
      height += w * (c + noise);
    }

    // Play with the contrast
    return pow(height, -2.);
}
```

Un cop aplicada aquesta nova distribució amb múltiples passades, podem generar imatges més properes als resultats que busquem.

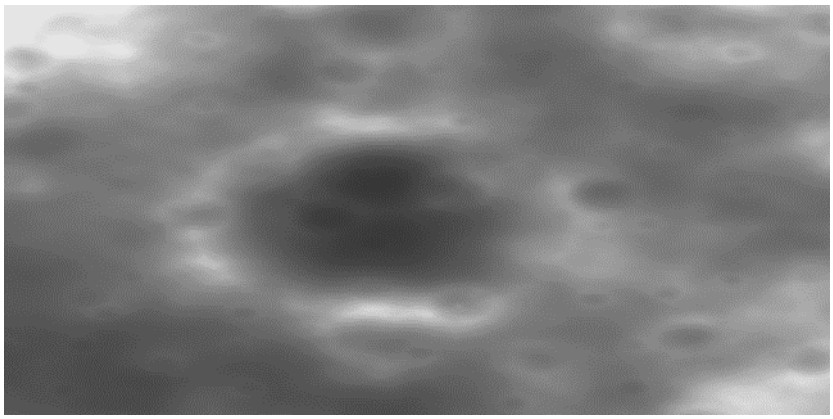

Figura 17 Resultat del fragmentShader amb una distribució de cràters. Font: pròpia

Per a poder projectar aquesta textura a la nostra geometria esfèrica, cal modificar el material que tenim (*NormalMaterial*) i fer ús d'un *shaderMaterial*. Al mateix constructor del material indicarem quin *fragmentShader* i *vertexShader* fem servir.

```
this.material =  new THREE.ShaderMaterial({
  fragmentShader: fragmentShader,
  vertexShader: vertexShader,
})
```

Finalment, cal importar els fitxers.

```
import fragmentShader from '../../shaders/fragmentShader.js'
import vertexShader from '../../shaders/vertexShader.js'
```

Ara podem veure la nostra esfera amb el nou material, encara que tenim alguns problemes, el primer és que tal com estan posicionats els vèrtexs de la nostra esfera, en projectar aquest mapa en una projecció cilíndrica, estira la textura en els seus pols.

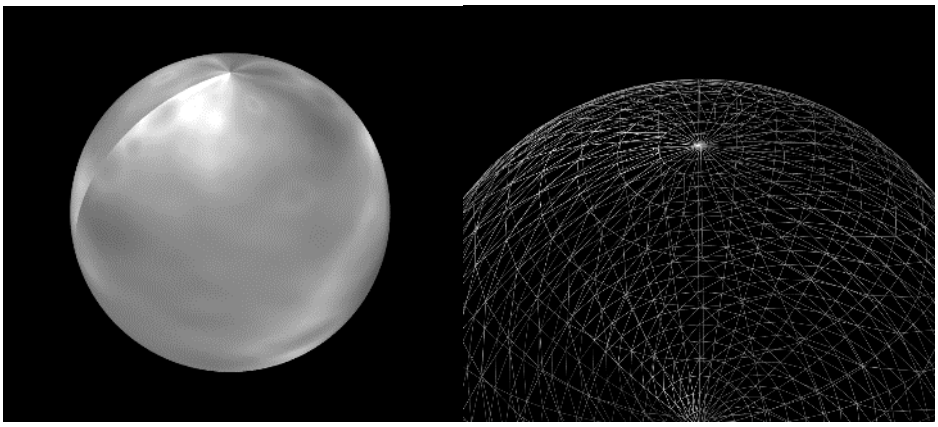

Figura 18 Shaders projectats a l'esfera i detall dels vèrtex. Font: pròpia

Per a solucionar això, podem fer servir una geometria de caixa i normalitzar els vèrtexs per formar una esfera, com podem veure a la Figura 19, la distribució de vèrtex és molt més homogènia.

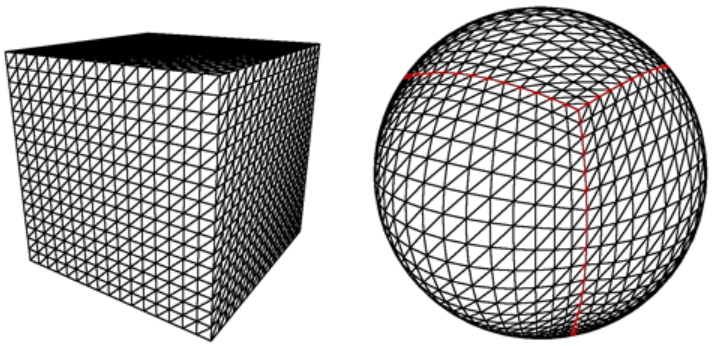

Figura 19 Cub *esferitzat*. Font: [44]

No obstant això, optarem per una solució més simple i ràpida, traduir les coordenades 2D (*vUv*) del nostre *shader* a coordenades cartesianes 3D, que ens donaran una latitud i una longitud determinades a l'esfera:

```
vec3 planeToCartesian(vec2 vUv) {
  float scale = 2.;
  float lat = 180. * vUv.y - 90.;
  float lon = 360. * vUv.x;

  return scale * vec3( sin(lon*PI/180.) * cos(lat*PI/180.), sin(lat*PI/180.),
cos(lon*PI/180.) * cos(lat*PI/180.));
}
```

Perquè els cràters tinguin un aspecte accidentat realista, fem una barreja amb un soroll artificial anomenat FBM (*Fractional Brownian Motion*) i s'escala de manera que els cràters més grans tinguin el màxim impacte sobre el terreny. El resultat és un mapa en escala de grisos de 8 bits amb valors del 0 (negre) al 255 (blanc) deformat a les porcions superiors i inferiors per adaptar-se a una projecció esfèrica.

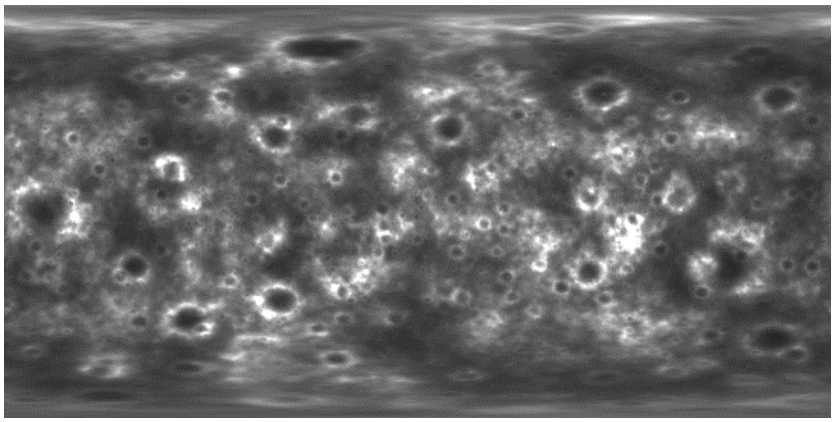

Figura 20 Mapa en escala de grisos generat proceduralment partir de shaders. Font: pròpia

Si aquest mapa el projectem a l'esfera, podem veure com hem solucionat el problema i ja no veiem cap imperfecció a la superfície.

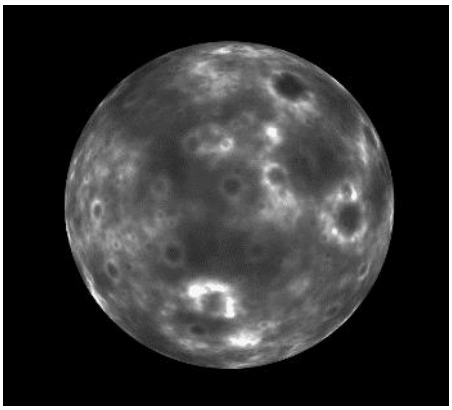

Figura 21 Mapa procedural projectat a l'esfera. Font: pròpia

2.2.3 MAPEIG DE TEXTURES (*UV MAPPING*)

L'acabem de veure en funcionament, el mapeig de textures és un mètode per agafar una textura bidimensional i mapejar-la en una geometria tridimensional. Imagineu un sistema de coordenades 2D a la part superior de la textura, amb (0,0) a la part inferior esquerra i (1,1) a la part superior dreta. Com que ja fem servir les lletres X, Y i Z per a les nostres coordenades 3D, ens referirem a la coordenada de la textura 2D amb les lletres U i V. D'aquí ve el nom de mapeig UV (*UV Mapping*).[9]

Aquesta és la fórmula utilitzada en el mapeig UV:

$$(u, v) \rightarrow (x, y, z)$$

(u,v) representa un punt de la textura, i (x,y,z) representa un punt de la geometria, definit a l'espai local. Tècnicament, un punt d'una geometria s'anomena vèrtex.[9]

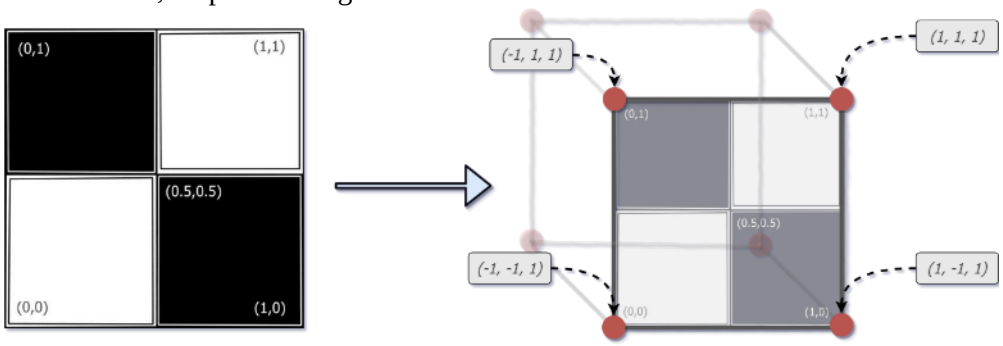

Figura 22 Diagrama del mapeig d'una textura a una geometria. Font: [9]

A la figura de dalt, la cantonada superior esquerra de la textura s'ha assignat a un vèrtex a la cantonada del cub amb coordenades (−1,1,1):

$$(0,1) \rightarrow (-1,1,1)$$

Es fan mapes similars per a les altres cinc cares del cub, donant lloc a una còpia completa de la textura a cadascuna de les sis cares del cub:

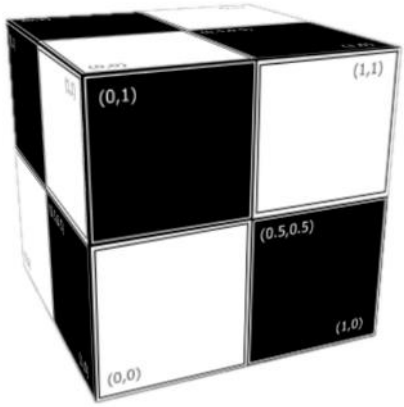

Figura 23 Mapeig d'una textura a un cub. Font: [9]

Tingueu en compte que no hi ha *mapeig* per al punt (0,5, 0,5), el centre de la textura. Només es *mapegen* les cantonades de la textura, a les vuit cantonades del cub, i la resta de punts s'endevinen a partir d'aquests. Per contra, un model complex ha de tenir moltes més coordenades UV definides per fer el *mapeig* de les parts de la textura als punts correctes de la geometria. [9]

2.2.4 IL·LUMINACIÓ BÀSICA

Per augmentar el realisme, podem afegir un punt de llum per simular una estrella i poder il·luminar el nostre planeta. Per a calcular com afectarà la il·luminació a cada píxel del nostre planeta, cal tenir en compte aspectes com el número i posició de les fonts de llum, les propietats del material, altres objectes, la posició de l'observador i el medi pel qual es propaga la llum.

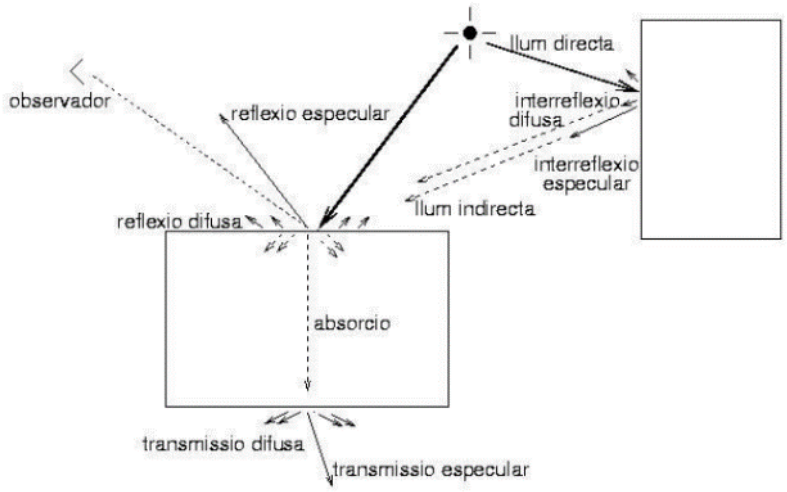

Figura 24 Il·luminació i elements que intervenen. Font: [45]

Existeixen diferents models d'il·luminació que simulen les lleis físiques que determinen el color d'un punt concret a la pantalla.

Model empíric ambient
En aquest model no es consideren punts de llum, és una il·luminació d'escena homogènia, per tant, tots els punts de l'escena reben la mateixa aportació de llum i s'observarà el mateix color en tots els punts d'un mateix objecte.

Aquest es pot representar com l'equació:

$$\boldsymbol{I_\lambda\,(P)\;=\;I_{a\lambda}\,k_{a\lambda}}$$

Fórmula (1)

$I_{a\lambda}$ És el color de la llum ambient
$k_{a\lambda}$ És el coeficient de reflexió ambient

Figura 25 **Model empíric ambient. Font: [45]**

Model empíric difús (Lambert)
En aquest model tenim un punts de llum concret que aplica en totes direccions, els objectes només tenen reflexió difusa pura i tenim un resultat més realista amb ombres encara que lluny de ser perfecte.

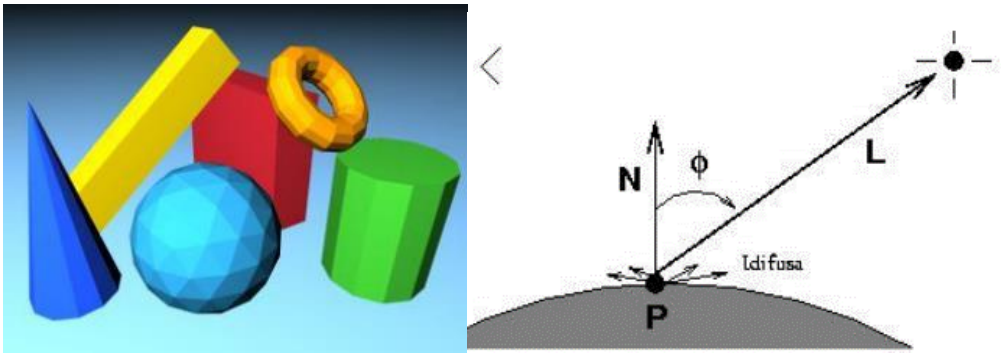

Figura 26 **Model empíric difús. Font: [45]**

Aquest es pot representar com l'equació:

$$\boldsymbol{I_\lambda(P)\;=\;I_{f\lambda}\,k_{d\lambda}\,cos(\Phi)=\;I_{f\lambda}\,k_{d\lambda}\,dot\,(N,L)}$$

$$si\;|\Phi|\;<\;90º$$

Fórmula (2)

$I_{f\lambda}$ És el color de la llum del focus puntual
$k_{d\lambda}$ És el coeficient de reflexió difusa del material
$cos(\Phi)$ És el cosinus de l'angle entre la llum incident i la normal a la superfície en el punt P. Aquest pot calcular-se com el producte escalar entre N i L si estan normalitzats.

Model empíric especular (Phong)
En aquest model tenim focus de llum puntuals i els objectes només tenen reflexió especular. Només es pot observar la reflexió especular en un punt si es troba en la direcció de la reflexió especular.[45]

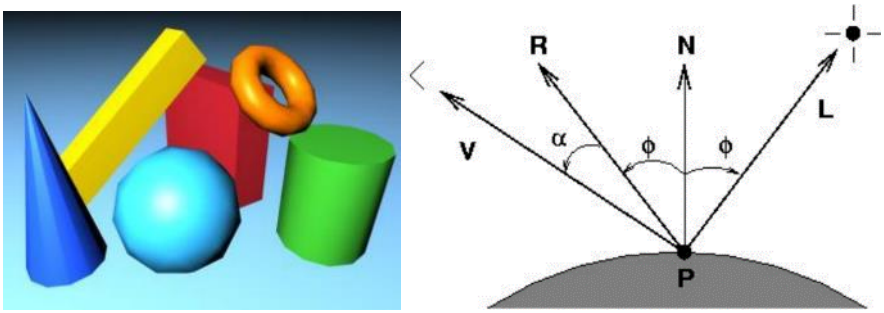

Figura 27 **Model empíric especular (Phong). Font: [45]**

Aquest es pot representar com l'equació:

$$I_\lambda(P) = I_{f\lambda}\, k_{s\lambda}\, cos^n(\alpha) = I_{f\lambda}\, k_{s\lambda}\, dot\,(R,v)^n \quad si\ |\Phi| < 90º$$

Fórmula (3)

La direcció d'especularitat és la simètrica de L respecte N i es pot calcular d'aquesta forma si tots els vectors són normalitzats:

$$\boldsymbol{R = 2n(n \cdot L) - L}$$

Fórmula (4)

$I_{f\lambda}$ És el color de la llum del focus puntual
$k_{s\lambda}$ És el coeficient de reflexió especular del material
n És l'exponent de reflexió especular
v És el vector normalitzat que uneix el punt P amb l'observador

Per sort, tot això ja ho incorpora la llibreria Three.js internament i nosaltres només hem d'escollir quin tipus de llum volem i segons els objectes i materials de l'escena, veurem un resultat més o menys realista. El problema és que el material que estem fent servir, el ShaderMaterial, no interactua amb elements externs com les llums que volem afegir, ja que s'han d'implementar internament al propi shader amb els càlculs que hem vist.

Per tant, en comptes de fer servir aquest material, farem ús del MeshStandardMaterial, que és un material que calcula la il·luminació basant-se en el model empíric especular de Phong[46] que hem vist i dona un resultat molt bo encara que sigui més car computacionalment. També ens dona les opcions de fer servir diferents mapes, que ens ofereix moltes més possibilitats, com veurem més endavant.

Afegir les llums és una tasca bastant simple, només cal declarar-les i afegir-les a l'escena. Afegim una llum direccional que actua com a estrella amb un color i intensitat concrets, després definim la seva posició indicant les coordenades en tres dimensions. També fem ús de llum ambient de baixa intensitat per donar un aspecte més realista.

```
this.ambientLight = new THREE.AmbientLight(0xffffff, 0.04);
this.scene.add(this.ambientLight);

this.directionalLight = new THREE.DirectionalLight( 0xffffff, 1.2 );
this.directionalLight.position.set( 1, 1, 0);
this.scene.add(this.directionalLight);
```

El material que farem servir és el *MeshStandardMaterial*, com hem comentat. Aquest té moltes propietats que podem definir, però de moment només li asignem un color.

```
this.material = new THREE.MeshStandardMaterial({
  color: new THREE.Color(0xFFFFFF)
});
```

El resultat és una escena amb il·luminació bàsica.

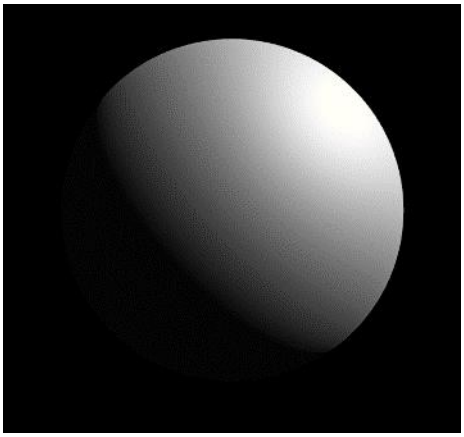

Figura 28 Esfera amb MeshStandardMaterial i il·luminació bàsica. Font: pròpia

2.2.5 MAPES DE TEXTURA

En aplicar aquest nou material, hem perdut el mapa o textura generat prèviament i en comptes d'això, tenim un planeta totalment blanc. Per a solucionar-ho, fem ús dels anomenats mapes, que és una funcionalitat que ens permet personalitzar aquest material de diferents formes. Per a generar aquest mapa que després apliquem al material, creem una nova escena temporal i càmera només per a aquest propòsit. L'escena és un únic pla amb el *shader* com a material.

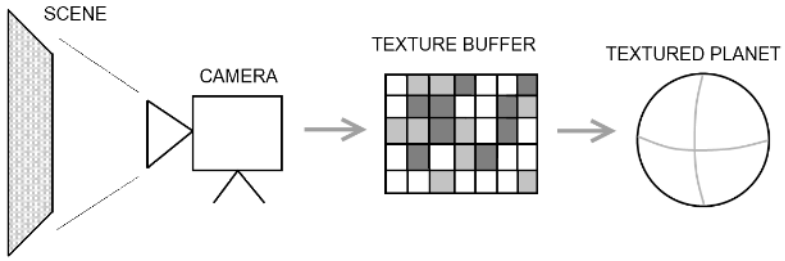

Figura 29 Flux de treball de l'escena a la textura. Font: [8]

La classe encarregada de fer això l'anomenem *Map*, i té la següent estructura:

```
class Map {

  setup() {
    let tempRes = 1000;
    this.texture = new THREE.WebGLRenderTarget(tempRes, tempRes, {minFilter:
THREE.LinearFilter, magFilter: THREE.LinearFilter, format: THREE.RGBAFormat});
    this.textureCamera = new THREE.OrthographicCamera(-tempRes/2, tempRes/2,
tempRes/2, -tempRes/2, -100, 100);
    this.textureCamera.position.z = 10;
    this.textureScene = new THREE.Scene();
    this.geo = new THREE.PlaneGeometry(1, 1);
    this.plane = new THREE.Mesh(this.geo, this.mat);
    this.plane.position.z = -10;
    this.textureScene.add(this.plane);
    this.map = this.texture;
  }

  render(props) {
    let resolution = props.resolution;
    this.texture.setSize(resolution, resolution);
    this.texture.needsUpdate = true;
    this.textureCamera.left = -resolution/2;
    this.textureCamera.right = resolution/2;
    this.textureCamera.top = resolution/2;
    this.textureCamera.bottom = -resolution/2;
    this.textureCamera.updateProjectionMatrix();

    this.geo = new THREE.PlaneGeometry(resolution, resolution);
    this.plane.geometry = this.geo;
    window.renderer.setRenderTarget(this.texture);
    window.renderer.render(this.textureScene, this.textureCamera);
    window.renderer.setRenderTarget(null);
    this.geo.dispose();
  }
}
```

Com podem veure, està formada per dues funcions principals, el *setup* simplement actua com a constructor i prepara tots els elements, la textura, el pla, la càmera, etc.

La funció *render* es crida cada cop que volem generar una nova textura, aquesta pot tenir diferents resolucions fàcilment configurant l'escena amb una mida o un altre. Això ens permetrà produir resultats de major qualitat a costa d'un major cost computacional, o reduir-lo per estalviar recursos i així adaptar-se al rendiment de cada equip.

Finalment, podem aplicar i renderitzar aquesta textura al material que fem servir.

```
this.material.map = this.textureMap.map.texture;

this.textureMap.render({
  resolution: this.resolution
});
```

El resultat és semblant al que teníem prèviament, abans de canviar de material del ShaderMaterial al StandardMaterial. Però ara tenim un sistema d'il·luminació molt precís i realista, i tots els avantatges que ens ofereix el StandardMaterial.

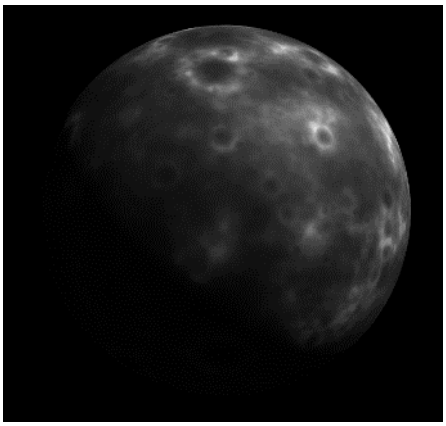

Figura 30 Resultat d'aplicar una textura al StandardMaterial. Font: pròpia

Ens apropem al resultat desitjat, però un planeta no és una esfera perfectament llisa. Aquí és on aprofitarem els mapes que podem implementar amb un StandardMaterial. Aquests ens permeten aplicar una sèrie de propietats al material. Fins ara hem creat el que seria un mapa de textura, però existeixen més, aquests són alguns a tindre en compte:

- alphaMap
  El mapa alfa és una textura en escala de grisos que controla l'opacitat a tota la superfície (negre: totalment transparent; blanc: totalment opac).

- bumpMap
  La textura per crear un mapa de desnivells. Els valors de blanc i negre es corresponen amb la profunditat percebuda en relació amb les llums. En realitat no afecta la geometria de l'objecte, només la il·luminació.

- displacementMap
  El mapa de desplaçament afecta la posició dels vèrtexs de la malla. A diferència d'altres mapes que només afecten la llum i l'ombra del material, els vèrtexs desplaçats poden projectar ombres, bloquejar altres objectes i actuar com a geometria real.

- normalMap
  Els valors RGB afecten la superfície normal de cada fragment de píxel i canvien la manera com s'il·lumina el color. Els mapes normals no canvien la forma real de la superfície, només la il·luminació.

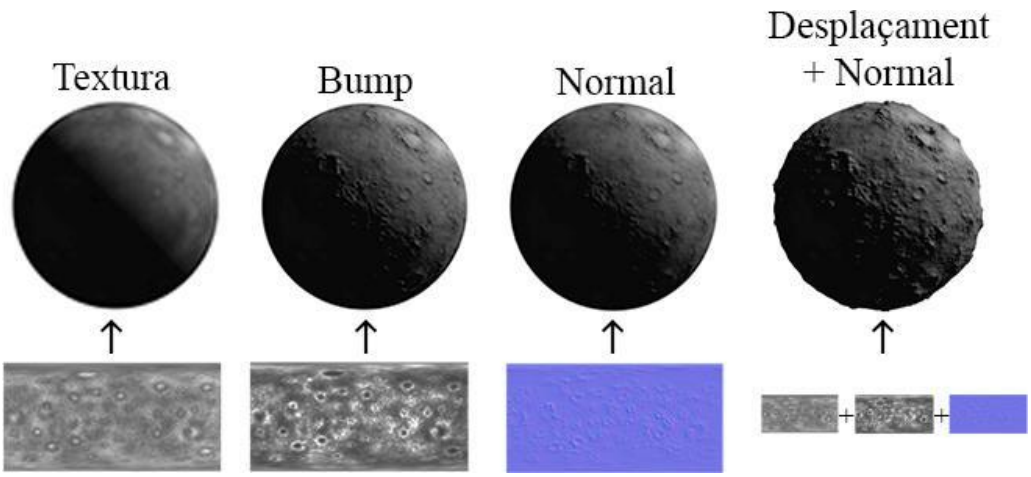

Figura 31 Comparació dels tipus de mapes aplicats al planeta. Font: pròpia

Els mapes *bump* són una manera fantàstica d'afegir detalls i realisme a un objecte renderitzat. Normalment, si volem petits detalls com esquerdes, arrugues, abollaments i línies fines, els hem de modelar amb molts polígons petits. Això redueix el rendiment perquè s'han de tenir en compte tots els polígons d'una escena durant la representació de cada fotograma. En canvi, aconseguirem el mateix efecte fent una mica de trampes.

El mapa *bump* és com un mapa de textures, però els valors no representen colors sinó desnivells i marques. Un valor més alt (més fosc) és un desnivell més destacat que un valor més baix (més clar). En lloc d'utilitzar aquest mapa directament durant la renderització de l'escena, és més fàcil convertir el mapa de desnivells a un mapa normal, on cada texel representa la normal de la superfície en aquest punt. Cada texel té tres components de color que contindran els valors x, y i z de la normal. Si les normals de la superfície apunten lluny de la font de llum, aquest punt serà més fosc. I si el normal apunta cap a la llum, aquest punt serà més clar. D'aquesta manera, semblarà que la superfície és irregular i detallada.[8].

Figura 32 Mapa normal a una esfera. Font: [8]

Per a crear un mapa *bump* bàsic, simplement cal pintar zones fosques i clares, ja que treballem en blanc i negre. Per això, una aproximació és utilitzar simplement el mapa de textures directament com a mapa *bump*.

Per crear un mapa normal, el que farem és convertir i reutilitzar el mapa de textura, pel que fa a la implementació, existeixen múltiples opcions.

Per exemple trobant els gradients en la direcció x i y per a cada texel, i calculant el vector ortogonal. Aquest vector serà la nova normal.

```
var pixel00 = new THREE.Vector3(0, 0, getHeight(x, y));
var pixel01 = new THREE.Vector3(0, 1, getHeight(x, y+1));
var pixel10 = new THREE.Vector3(1, 0, getHeight(x+1, y));
var orto = pixel10.sub(pixel00).cross(pixel01.sub(pixel00)).normalize();
normalMap[i+0] = (orto.x/2+0.5)*255;
normalMap[i+1] = (orto.y/2+0.5)*255;
normalMap[i+2] = (orto.z/2+0.5)*255;
```

El mapa normal RGB generat serà com el de la **¡Error! No se encuentra el origen de la referencia.**. Quedant clarament denotats els desnivells i pertubacions del terreny, que quedaran reflectits en fer el renderitzat de l'escena.

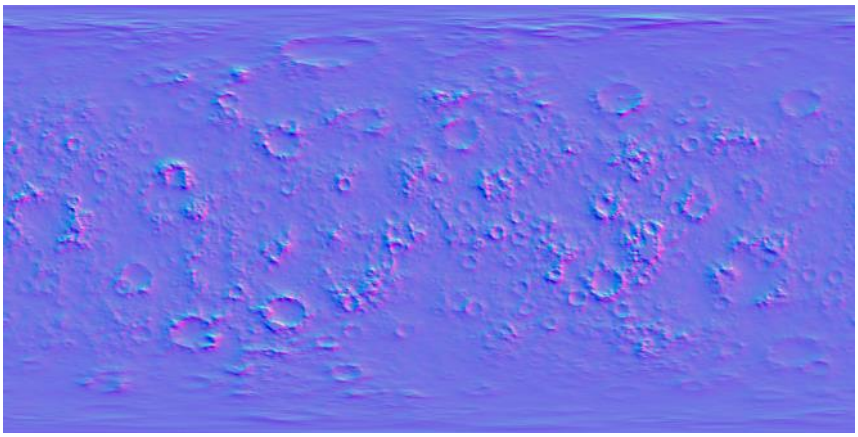

Figura 33 Mapa normal generat a partir del mapa de textura principal. Font: pròpia

Podem crear classes per a cada un d'aquests mapes de forma que tindrem un major control sobre ells i una millor estructura. L'estructura a seguir serà una classe que estendrà a la prèviament creada classe *map* amb un constructor i un parell de mètodes simples. Setup crea el material amb els *shaders* corresponents i es cridarà a render cada cop que es creï o s'actualitzi el *map*.

```
class TextureMap extends Map {
  constructor() {
    super();
    this.setup();
    super.setup();
  }

  setup() {
    const uniforms = {
      u_resolution: { value: { x: null, y: null } },
      u_time: { value: 0 },
      u_mouse: { value: { x: null, y: null } },
    }

    this.mat = new THREE.ShaderMaterial({
      uniforms: uniforms,
      vertexShader: vertShader,
      fragmentShader: fragShader,
      transparent: true,
      depthWrite: false
    });
  }

  render(props) {
    this.mat.uniforms.u_time.value = props.time;
    this.mat.needsUpdate = true;
    super.render(props);
  }
}
```

En carregar el material ens hem d'assegurar d'assignar tots els *maps* que fem servir així com l'escala d'aquests, que ens permeten modificar l'impacte que té cada mapa sobre el material en l'escena final.

```
this.material.map = this.textureMap.map.texture;

this.material.displacementMap = this.heightMap.map.texture;
this.material.displacementScale = this.displacementScale;

this.material.normalMap = this.normalMap.map.texture;
this.material.normalScale = new THREE.Vector2(this.normalScale, this.normalScale);
```

Un cop implementat, el planeta resultant és semblant a com es pensa que va ser la Terra en el seu període de formació inicial. Com que llavors les temperatures eren encara molt altes, com hem explicat, podem afegir una tonalitat vermella al resultat com a una primera aproximació per tractar de representar aquesta etapa (en els capítols següents veurem com millorar-ho).

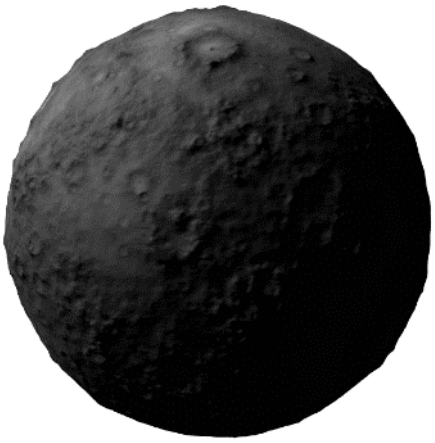

Figura 34 Resultat final amb aparença semblant a la Terra jove. Font: pròpia

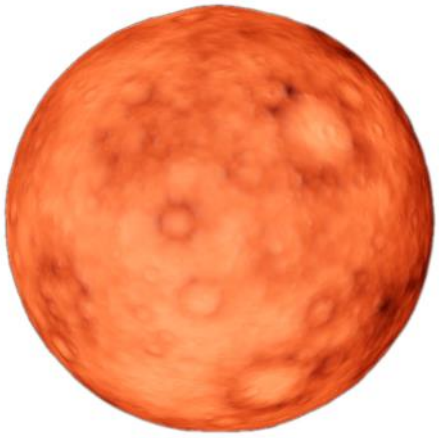

Figura 35 Resultat final amb una tonalitat vermellosa. Font: pròpia

CAPÍTOL 3. OCEANS I EVOLUCIÓ TEMPORAL

En aquest capítol es presenta l'origen de l'aigua, el procés de formació d'oceans i com implementar el pas del temps en la simulació per aconseguir recrear aquests efectes sobre la superfície del planeta. Apareixen elements clau com els *buffers*, que ens permeten afegir complexitat al programa.

3.1 ORIGEN DELS OCEANS

Durant els pròxims centenars de milions d'anys (4500 a 3900)[3], l'oceà de magma es va refredar prou per formar una superfície sòlida. Diversos processos van ocórrer en aquest període, entre ells, l'aparició de l'aigua repartida pel planeta en forma d'oceans[11].

Es creu que part d'aquesta aigua va ser lliurada pels cometes i meteorits que s'estavellaven a la superfície[11] i la resta és aigua que va quedar atrapada durant la formació del planeta.[28] Gràcies a la baixada de temperatura de la Terra, es van començar a formar núvols que finalment dipositaven encara més aigua a la superfície[25].

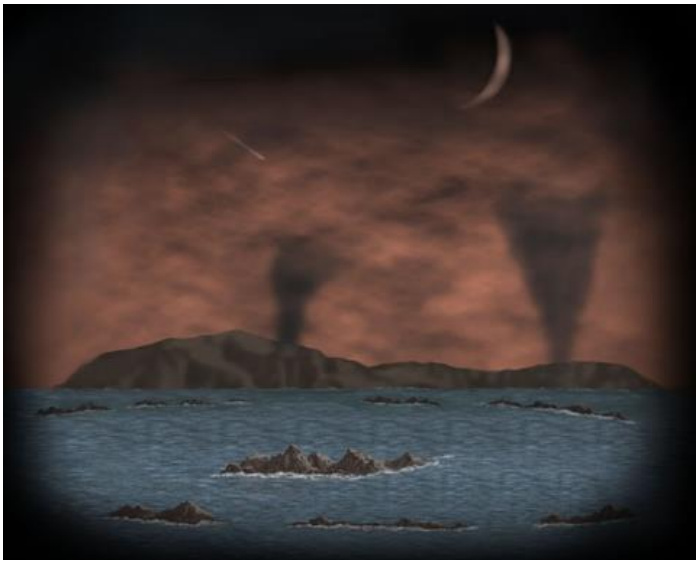

Figura 36 Representació artística dels primers oceans a la Terra. Font: [29]

3.2 IMPLEMENTACIÓ

Per introduir aquests oceans en el nostre programa, modificarem els fitxers de shaders que tenim, ja que partim del terreny que hem generat. Aprofitem el mateix mapa de textura per obtenir un altre mapa en escala de grisos que farem servir per determinar l'elevació del terreny, de forma que les àrees més baixes podrem omplir les d'aigua.

El primer que necessitarem serà crear un altre fragment *shader* que gestioni tots els altres shaders que tinguem, ja que estem complicant el programa i caldrà treballar amb *buffers*.

Els *buffers* son objectes de OpenGL que ens permeten guardar i recuperar dades. Això ho farem servir per poder executar multiples *shaders* diferents i permitir la comunicació entre ells.

Per acomodar aquesta nova estructura, caldrà modificar les classes *JavaScript* que tenim, però el funcionament és el mateix.

Tindrem un parell de *buffers*, el principal, que, entre altres coses, coordinarà els altres *buffers* i també tenim l'encarregat de generar el terreny.

```
// Main buffer
this.bufferMain = new BufferShader(
  VERT,
  BUFFER_MAIN_FRAG,
  {
    uTime: { value: 0 },
    uFrame: { value: 0 },
    uResolution: { value: this.resolutionVector },
    iChannel0: { value: null }
  });


// Geo buffer
this.bufferGeo = new BufferShader(
  VERT,
  BUFFER_GEO_FRAG,
  {
    uTime: { value: 0 },
    uFrame: { value: 0 },
    uResolution: { value: this.resolutionVector },
    iChannel0: { value: null }
  });
```

Podeu veure que indiquem un paràmetre anomenat **iChannel0**, aquest és un *uniform* que farem servir per enviar les dades. Com que el que enviem són resultats d'execucions dels *shaders*, és a dir una textura, caldrà fer ús de la funció que ens proporciona GLSL anomenada **texture**. Això ens permet obtenir el valor de la textura lligada al *buffer* que indiquem en la crida a la funció. Recordem que aquests shaders s'executen per cada píxel de la pantalla i **p** representa el píxel actual, per tant, estem obtenint el píxel corresponent en el *buffer* indicat.

```
#define buf(p) texture(iChannel0,(p)/uResolution.xy)
```

Com que farem servir més vegades aquesta funció, és bona idea referenciar-la sota un nom com *buf* per simplificar el codi. En fer la crida de la funció, rebem un vector de 4 dimensions.

Per guardar el mapa d'altura, al *shader* de generació del terreny, podem fer ús de la dimensió z del vector de retorn.

```
gl_FragColor.z = max(gl_FragColor.z, 0.);
```

Ara al shader principal podrem recuperar aquest valor amb aquesta crida:

```
float y = buf(p).z;
```

Si executem el shader i dibuixem aquest valor, podrem veure un resultat com aquest:

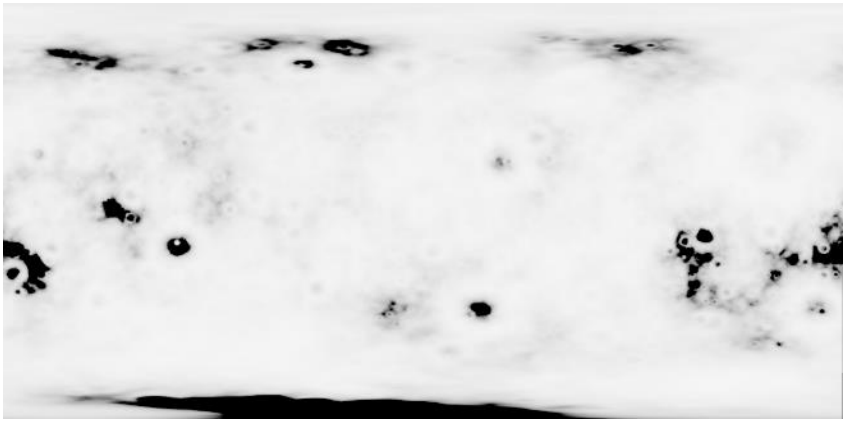

Figura 37 Mapa d'altitud, valors més clars indiquen terreny elevat i els foscos son més baixos. Font: pròpia

El color de l'aigua el podem definir amb dos colors, *deepWaterColor*, que fa referència al color de l'aigua més profunda, per tant, amb un color més fosc, i *shallowWaterColor* que és per a l'aigua més propera a la costa. El factor que decidirà si mostrar un color més proper a un o a l'altre el defineix el mapa d'altura que tenim guardat a la variable **y**, per tant, podem fer ús de la funció **mix** per, com indica el nom, barrejar els colors en funció de l'altura.

```
vec4 deepWaterColor = vec4(0.01, 0.02, 0.08, 1);
vec4 shallowWaterColor = vec4(0.11, 0.28, 0.51, 1);
vec4 ocean = mix(deepWaterColor, shallowWaterColor, y);
```

Si visualitzem també aquest mapa, podem veure que és pràcticament idèntic que el mapa d'altitud amb diferents colors.

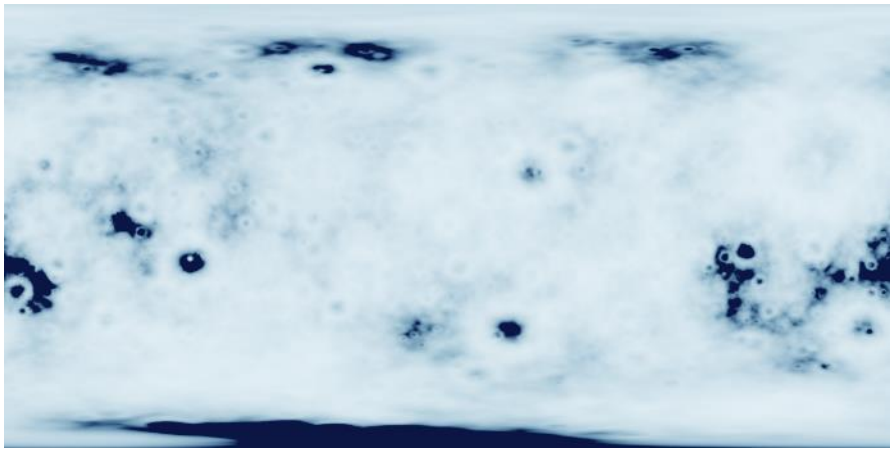

Figura 38 Mapa de l'oceà. Font: pròpia

Per a combinar el mapa de l'oceà i el del terreny, podem fer-ho fàcilment tenint en compte el valor d'altura del píxel actual i llavors decidir si dibuixar part del mapa de l'oceà o el mapa del terreny.

El codi per fer això és el següent. En aquest cas, si l'altura no arriba al 7, es dibuixa part de l'oceà i en cas contrari es dibuixa el terreny. Podem jugar amb aquest valor per veure com impacta al resultat final.

```
if (y < 7.) {
    r = ocean;
} else {
    r = land;
}
```

Si executem el programa, veurem un resultat com aquest:

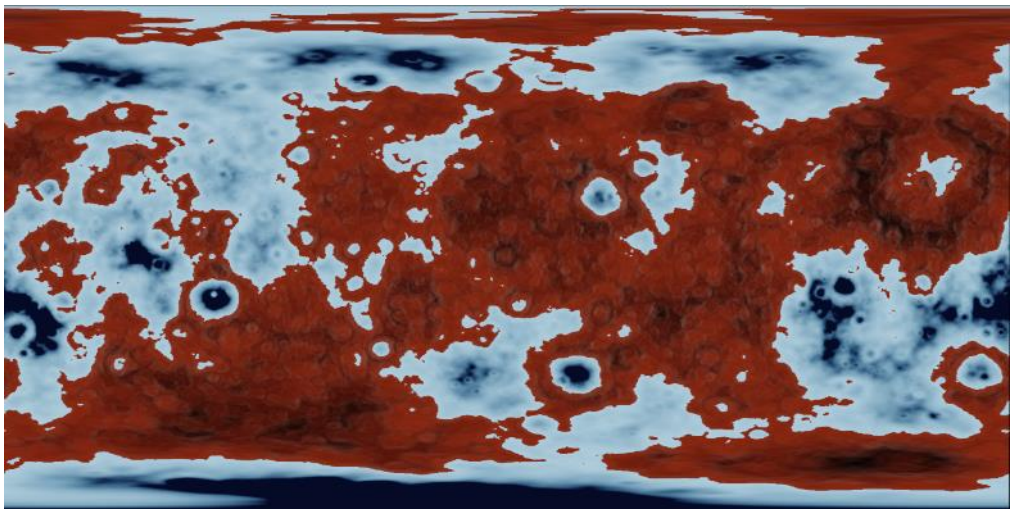

Figura 39 Mapa de terreny amb oceà. Font: pròpia

3.3 AFEGINT MOVIMENT

Fins ara hem vist com podem implementar elements individualment, però encara es poden millorar molt els resultats. Si volem mostrar una evolució en el temps, haurem de ser capaços d'introduir en aquests càlculs una variable de temps i executar en temps real, múltiples vegades per segon, aquests *shaders* per formar una espècie d'animació.

Guardarem una variable que s'anirà incrementant mentre el programa estigui en execució, aquesta l'enviarem als *shaders* mitjançant un *uniform* anomenat uTime.

També caldrà definir els períodes on s'executarà cada un dels *shaders* al programa. Això ho podem fer en un nou fitxer que agrupi constants com aquestes, codi que es fa servir en tots els altres *shaders*, etc..

El codi que podem incloure és el següent:

```
#define PI 3.14159265359

// Events
#define OCEAN_START_TIME 15.
#define LAND_START_TIME 20.
#define OCEAN_END_TIME 25.
#define LAND_END_TIME 30.

// Funcions
...
```

Aquests valors són orientatius, però de moment com volem que l'oceà aparegui un cop ha avançat uns segons la simulació, (en aquest cas són concretament 15s) caldrà fer ús d'aquests nous valors que hem definit. Per això podem redefinir el codi del *shader* principal on dibuixem la terra i oceà en funció de l'altura.

```
if (y < OCEAN_DEPTH && uTime > OCEAN_START_TIME) {
    r = mix(land, ocean, smoothstep(0., 2., uTime - OCEAN_START_TIME));
} else {
    r = land;
}
```

Prèviament, hem vist com funciona la funció mix, fent una barreja dels primers dos paràmetres controlada pel tercer. Aquí apareix una funció ***smoothstep***, que el que fa és retornar un valor suavitzat segons la funció definida, en aquest cas en funció del temps. La gràfica que dibuixa és la següent:

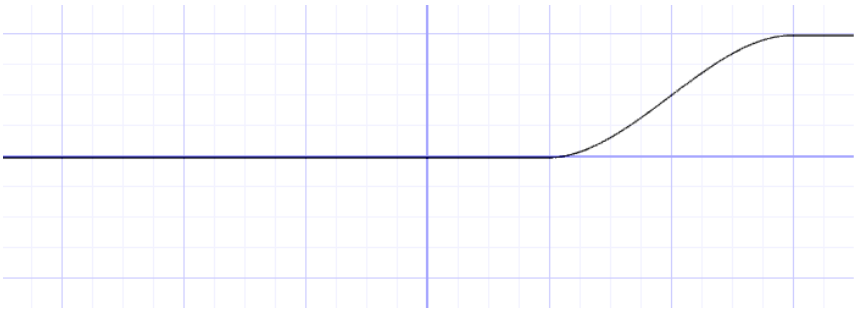

Figura 40 Funció smoothstep. Font: pròpia

Amb això aconseguim una transició suau i homogènia entre els diferents estats de la simulació.

En executar el programa, el planeta comença en el seu estat de formació inicial i a mesura que arriba al temps establert de 15 segons, comencen a aparèixer petits oceans d'aigua que van omplint el planeta. El resultat és el següent:

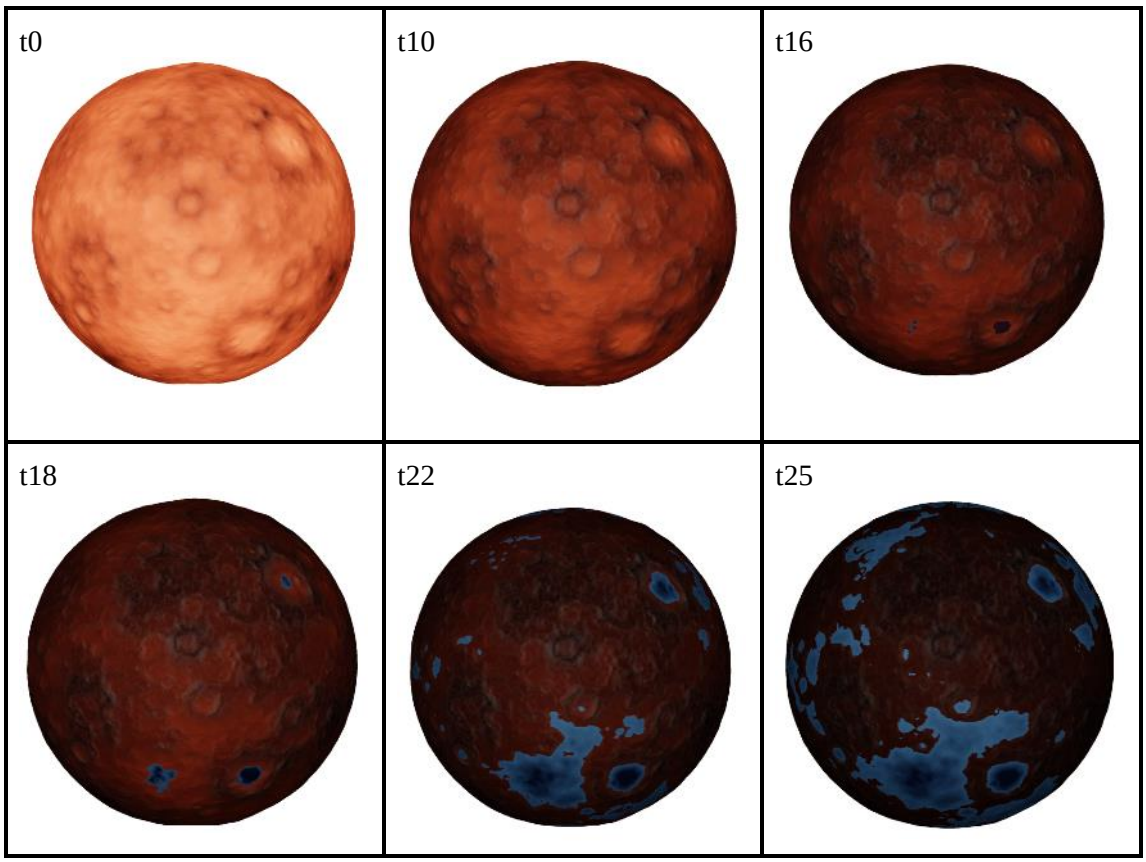

Figura 41 Transició del terreny amb oceans en executar el programa. Font: pròpia

CAPÍTOL 4. RIUS I EROSIÓ HIDRÀULICA

En aquest capítol es presenta un nou procés per millorar la generació del terreny per aconseguir una aparença accidentada natural en el terreny. Es tracta del procés de formació de rius i que causa que l'aigua erosioni el terreny, un procés propi del desenvolupament natural del planeta.

4.1 QUE ES L'EROSIÓ HIDRÀULICA

L'erosió hidràulica és un procés geològic pel qual l'aigua transforma el terreny al llarg del temps[30]. Es produeix pel pas de l'aigua en moviment que va retirant material de la terra, és a dir que produeix l'erosió del sòl i el va desgastant a poc a poc, i després diposita aquest material aigües avall. [31]

El material arrossegat pot comprendre des de fines partícules sorrenques a roques de grans dimensions i és un procés que contribueix de forma substancial al modelatge del terreny, tant a la zona de càrrega de material com a la zona de dipòsit. [31]

Quan un riu entra en aigua estancada, la seva velocitat disminueix fins a aturar-se. El rierol es mou d'anada i tornada per la regió. La riera deixa caure els seus sediments en un ampli dipòsit de forma triangular anomenat delta[34].

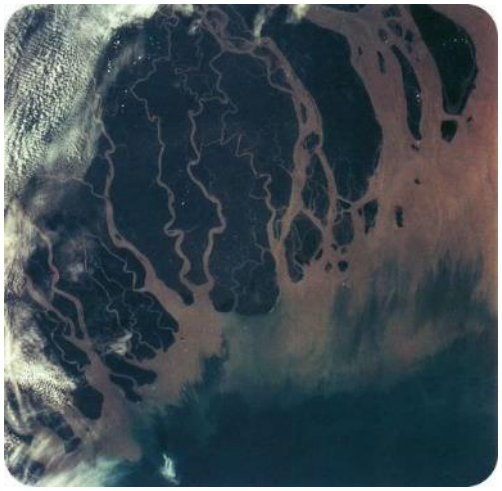

Figura 42 El riu Ganges forma un delta a BanglaDesh. Font: [34]

Els cursos d'aigua fluvial són un tipus d'erosió hidràulica on els rius i altres corrents més petits són els responsables de modelar el terreny des de les altes muntanyes fins a les zones costaneres, passant per una immensa varietat de formes intermèdies. En general tenen forma de V a les zones més escarpades, és a dir, a les zones muntanyoses, mentre que tendeixen a eixamplar-se a les zones més planes. A més, a les zones muntanyoses sol arrossegar roques més grans. [31]

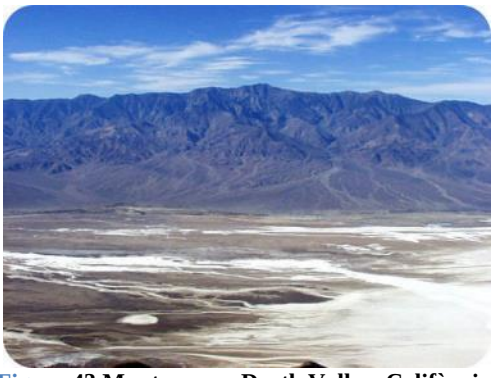

Figura 43 Muntanyes a Death Valley, Califòrnia. Font: [34]

Figura 44 Un curs d'aigua fluvial. Font: [31]

Existeixen més processos d'erosió específics, cada un amb unes característiques concretes i un funcionament diferent.

4.2 TRANSFORMANT EL TERRENY

Hi ha diverses formes d'enfocar el problema quan es tracta de simular l'erosió. Tots els mètodes simulen el mateix fenomen: l'aigua es mou de llocs alts a llocs baixos, erosiona el terreny a mesura que flueix i diposita sediments a mesura que van més enllà dels seus camins.

Aquest procés dona lloc a una sèrie de característiques del terreny reconeixibles, com ara barrancs i valls on flueixen els rius, deltes on es troben amb el seu destí i ventalls al·luvials on els rierols més petits es combinen en rius més grans.

L'objectiu d'aquest mètode és crear entorns creïbles en lloc d'assolir un alt grau de realisme. La fidelitat es pot sacrificar per la velocitat, sempre que els resultats semblin naturals.

El primer que farem serà iniciar el mapa en el qual guardem aquesta informació amb valors aleatoris calculats per una funció de *hash* per aconseguir resultats homogenis al terreny. Fem servir la **dimensió w** del vector de sortida per aquesta tasca.

```
gl_FragColor.w = hash12(p);
```

Hi ha una varietat de simulacions de flux d'aigua disponibles per a aquesta tasca, però una dificultat aquí és que la resolució del mapa del terreny és bastant baixa per a un planeta sencer.

Per tant, el model haurà de ser capaç de simular rius que no tenen més d'un píxel d'amplada. Richard Barnes proposa un model senzill que aconsegueix això en una aportació anomenada **acceleració d'un model d'incisió fluvial i d'evolució del paisatge amb paral·lelisme**.[32]

Cada píxel examina els seus vuit veïns per determinar quina direcció té la major disminució d'elevació (ajustada pel fet que les diagonals veïnes estan més lluny). Aquesta direcció de major pendent és on viatjarà l'aigua que surt d'aquest píxel. L'aigua primer es distribueix per pluja, i després es transporta entre píxels veïns amb el pas de temps.

```
// Direcció del flux d'aigua en el punt p
vec2 rec(vec2 p) {
    vec2 d = vec2(0);
    if (slope(p + N,  p) >= slope(p + d, p)) d = N;
    if (slope(p + NE, p) >= slope(p + d, p)) d = NE;
    if (slope(p + E,  p) >= slope(p + d, p)) d = E;
    if (slope(p + SE, p) >= slope(p + d, p)) d = SE;
    if (slope(p + S,  p) >= slope(p + d, p)) d = S;
    if (slope(p + SW, p) >= slope(p + d, p)) d = SW;
    if (slope(p + W,  p) >= slope(p + d, p)) d = W;
    if (slope(p + NW, p) >= slope(p + d, p)) d = NW;
    return d;
}

// Pendent entre dos punts
float slope(vec2 p, vec2 q) {
    if (p == q) return 0.;
    return (buf(q).z - buf(p).z) / distance(p,q);
}
```

Si trobem que no hi ha cap pendent, fem que es dipositin els sediments, i en cas que hi hagi un pendent, calculem la quantitat d'erosió que ocorre.
Aquest valor ens el dona la llei de potència del corrent, que prediu la velocitat d'erosió d'un riu al seu llit.

$$E = KA^m S^n$$

Fórmula 5

On E és l'erosió causada, A és la quantitat d'aigua en aquest punt i S és el pendent del canal.[33]

Els paràmetres K, m i n no són necessàriament constants, sinó que poden variar en funció de les lleis d'escala suposades, el procés d'erosió, l'erodibilitat de la roca, el clima, el flux de sediments i/o el llindar d'erosió. No obstant això, les observacions de l'escala hidràulica de rius reals indiquen que la relació m/n hauria d'estar al voltant de 0,5[33]. Això transformat a codi resulta en aquesta línia, però encara està incomplet.

```
elevació -= 0.05 * pow(aigua, 0.8) * pow(pendent, 2.);
```

El resultat del càlcul serà la distancia que hem de reduir en aquest punt. Com que aquesta es guarda en la dimensió z, com hem indicat, serà finalment on s'apliqui la modificació.

```
vec4 receiver = buf(p + rec(p));
float pslope = (gl_FragColor.z - receiver.z) / length(rec(p));
float dz = min(pow(floor(gl_FragColor.w), 0.8) * pow(pslope, 2.), gl_FragColor.z);
gl_FragColor.z = max(gl_FragColor.z - 0.05 * dz, receiver.z);
```

D'aquesta forma tenim l'elevació i la quantitat d'aigua situada al punt actual, juntament amb el pendent en la direcció en què va l'aigua. La disminució d'elevació es limita de manera que no sigui inferior a la ubicació on flueix l'aigua.

La interacció entre l'aigua i la pròpia erosió dona lloc a la formació natural de conques fluvials al terreny:

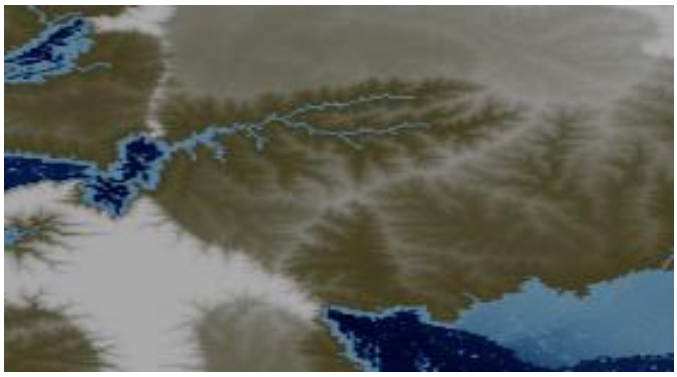

Figura 45 **Erosió en funcionament. Font: pròpia**

4.3 VISUALITZACIÓ DE CONQUES HIDROGRÀFIQUES

Si acolorim els canals connectats de forma que el mateix color representi la ubicació de la desembocadura del riu, és possible produir visualitzacions que recorden els mapes de conques fluvials reals:

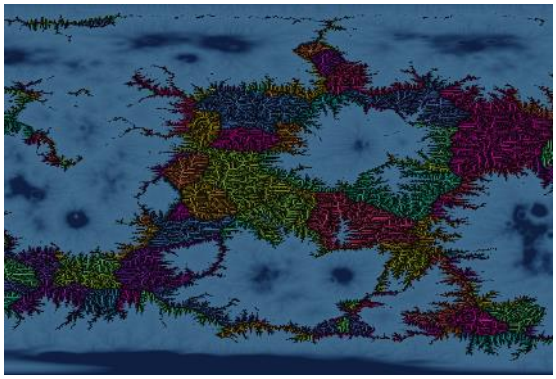

Figura 46 Conca hidrogràfica acolorida en el programa. Font: pròpia

Figura 47 Conca hidrogràfica real d'Europa. Font: [35]

Per poder visualitzar aquests rius, primer cal guardar la part fraccionària del valor del punt actual en la **dimensió w**. Això ho podem fer amb la funció de GLSL anomenada **fract**. Per cada una de les 8 direccions, afegirem el valor mínim entre el valor actual i el que trobem a la direcció a consultar, ja que busquem recórrer les bifurcacions dels rius.

```
gl_FragColor.w = 1. + fract(gl_FragColor.w);
if (rec(p + N)  == -N)  gl_FragColor.w += floor(buf(p + N).w);
if (rec(p + NE) == -NE) gl_FragColor.w += floor(buf(p + NE).w);
if (rec(p + E)  == -E)  gl_FragColor.w += floor(buf(p + E).w);
if (rec(p + SE) == -SE) gl_FragColor.w += floor(buf(p + SE).w);
if (rec(p + S)  == -S)  gl_FragColor.w += floor(buf(p + S).w);
if (rec(p + SW) == -SW) gl_FragColor.w += floor(buf(p + SW).w);
if (rec(p + W)  == -W)  gl_FragColor.w += floor(buf(p + W).w);
if (rec(p + NW) == -NW) gl_FragColor.w += floor(buf(p + NW).w);
```

Fem servir aquesta funció per mostrar una barreja del mapa de terreny i dibuixar per sobre els rius acolorits. Aquesta informació la consultem directament de la **dimensió w** del vector.

```
vec4 map_rivers(vec2 fragCoord) {
    vec4 fragColor = map_land(fragCoord, true);
    float flow = buf(fragCoord).w;
    fragColor.rgb = mix(fragColor.rgb, .6 + .6 * cos(2.*PI * fract(flow) +
vec3(0,23,21)), clamp(0.15 * log(floor(flow)), 0., 1.));
    return fragColor;
}
```

Amb això podem produir representacions molt visuals del funcionament d'aquest fenomen en la simulació.

CAPÍTOL 5. MODEL TECTÒNIC

En aquest capítol es presenta un nou procés geogràfic que deriva en el que coneixem com plaques tectòniques. S'explica l'existència de les plaques que formen la superfície de la Terra i als desplaçaments que s'observen entre elles en el seu moviment sobre el planeta. Finalment, es veu el funcionament de la implementació en forma de codi amb l'objectiu d'aconseguir una aproximació realista recreant les principals característiques.

5.1 ELS PRIMERS CONTINENTS

Durant els pròxims centenars de milions d'anys (4500 a 3900)[3], l'oceà de magma de la Terra es va refredar prou per formar una superfície sòlida, l'atmosfera terrestre es va començar a formar gràcies a les erupcions volcàniques, així com l'aigua i altres gasos lliurats pels cometes i meteorits que s'estavellaven a la superfície. Aquest va ser el primer pas cap al desenvolupament de les plaques tectòniques del nostre planeta[11].

Aquestes es mouen lentament per la superfície de la Terra durant centenars de milions d'anys; Es creu que van començar a moure's lentament fa 3.400 milions d'anys[37]. El moviment relatiu d'aquestes plaques era molt més elevat que en l'actualitat[38], ara per ara oscil·la entre zero i 10 cm anualment.[37]

El procés que impulsa les plaques tectòniques és la convecció, el resultat del flux de calor des de l'interior de la Terra a la superfície terrestre. Implica la creació de plaques tectòniques rígides a les dorsals oceàniques. Durant aquest període, la temperatura del mantell era propera als 1.600 °C, de manera que la convecció ocorria de forma més ràpida. És probable que les zones de subducció fossin més comunes i, per tant, les plaques tectòniques eren més petites.[26].

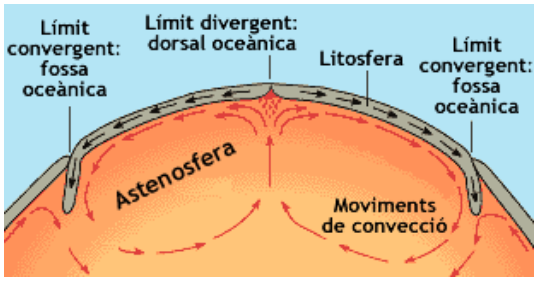

Figura 48 El motor principal de les plaques tectòniques, els corrents de convecció. Font: [40]

Els límits són les vores d'una placa i és on es presenta la major activitat "tectònica" (sismes, formació de muntanyes, activitat volcànica), ja que és en aquestes zones on es produeix la interacció entre plaques[39]. Existeixen tres tipus de límit:

- **Divergents**
  En els límits divergents dels oceans, el magma de les profunditats del mantell terrestre s'eleva cap a la superfície i separa dues o més plaques. Muntanyes i volcans s'aixequen al llarg de l'àrea. A terra, es formen canals gegants on les plaques es separen. Si les plaques continuen divergent, poden arribar a separar continents per formar una nova massa terrestre. Aleshores, una dorsal oceànica marcaria el límit entre les plaques.[39]

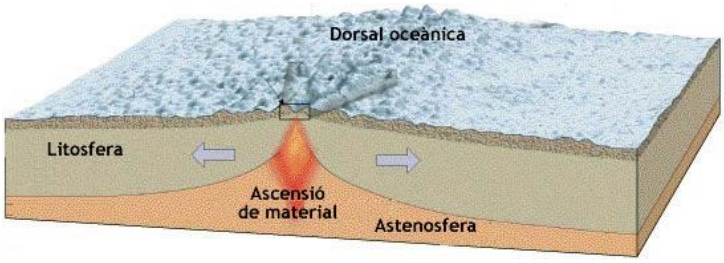

Figura 49 Límit divergent entre dues plaques. Font: [40]

- **Convergents**
  Allà on les plaques xoquen, l'escorça s'ensorra i es doblega en cadenes muntanyoses. També es produeixen quan una placa oceànica s'enfonsa, en un procés anomenat subducció, sota una massa terrestre. A mesura que la placa superior s'aixeca, també forma serralades. A més, la placa que s'enfonsa es fon i sovint genera erupcions volcàniques.[39]

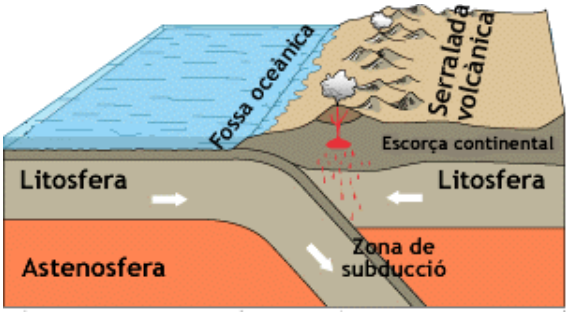

Figura 50 **Límit convergent entre una placa oceànica i una continental. Font: [40]**

- **Transformants**
  Tenen lloc a zones on dues plaques ni s'ajunten ni se separen sinó que es mouen lateralment, lliscant entre elles. Aquests límits no estan associats a l'aparició de volcans, però sí a terratrèmols.

  L'exemple més conegut és el de la falla de San Andreas, a Califòrnia, responsable dels terratrèmols que es produeixen a la ciutat de San Francisco. Aquesta falla té uns 1.300 km de longitud i es mou uns 5 cm per any de mitjana.[40]

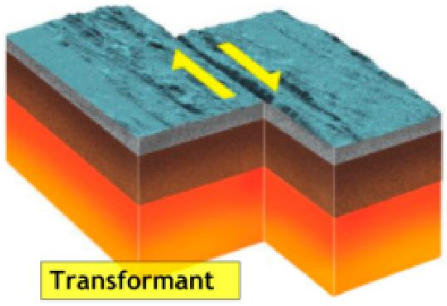

Figura 51 **Límit transformant entre dues plaques. Font: [27]**

5.2 IMPLEMENTACIÓ

Per poder simular, la formació de muntanyes, trinxeres oceàniques i formes de terra continentals, necessitem implementar un model de moviment tectònic amb el qual puguem representar aquests fenòmens en el nostre programa.

El que fem és generar de forma aleatòria la ubicació d'aquestes plaques, que estan repartides sobre el terreny. Per això fem servir una funció *hash* quin sigui el valor de retorn, decidim si es crearà una nova placa tectònica en aquest punt i també la quantitat i mida que tindran. En aquest cas fem servir la dimensió x del vector de sortida per guardar aquest nou mapa. Tots els píxels d'una placa emmagatzemen la velocitat del moviment de la placa.

```
if (length(hash33(vec3(p,uFrame))) < 0.006)
   gl_FragColor.x = fract(hash13(vec3(p,uFrame)) + 0.25);
```

Aquestes plaques creixen de mida a mesura que passa el temps amb un model d'agregació molt senzill, que selecciona aleatòriament els punts veïns i els afegeix a una placa si encara no s'han assignat a una altra placa.

```
int dir = int(4.*hash13(vec3(p,uFrame)));
switch (dir) {
    case 0: gl_FragColor.x = s.x; break;
    case 1: gl_FragColor.x = w.x; break;
    case 2: gl_FragColor.x = n.x; break;
    case 3: gl_FragColor.x = e.x; break;
}
```

Simular el moviment continu de les plaques és una tasca difícil, ja que requeriria tenir en compte els moviments mesurats en fraccions de píxel. Per evitar-ho, les plaques es mouen en intervals de temps discrets, per un píxel sencer, ja sigui horitzontalment o verticalment. Aquests temps són aleatoris per a cada placa de manera que la velocitat mitjana es mantingui a la velocitat i la direcció establertes, i també de manera que és poc probable que les plaques veïnes es moguin simultàniament. La funció encarregada de definir el moviment de les plaques pel terreny en una direcció concreta és la següent:

```
vec2 plate_move(float q, float uFrame, float uTime) {
    // Fora del rang de temps no hi ha moviment
    if (uTime >= TECTONICS_END_TIME && uTime < END_TIME) return vec2(0);

    // Obtenir un valor aleatori entre (0,0) i (1,1)
    vec2 v = vec2(cos(2.*PI*q), sin(2.*PI*q));

    // Possibilitats de que es mogui en aquest frame
    if (hash13(vec3(v,uFrame)) < 0.05) {

        // Decideix la direcció del moviment
        if (hash13(vec3(v+1.,uFrame)) < abs(v.x) / (abs(v.x) + abs(v.y))) {
            return vec2(sign(v.x),0.);
        } else {
            return vec2(0.,sign(v.y));
        }
    }
    return vec2(0);
}
```

Per a poder moure les plaques de forma homogènia, els píxels que pertanyen a cada placa consulten a quina direcció han de moure's.

```
// Si el moviment es cap al sud
} else if (move(n.x) == S) {
    if (move(gl_FragColor.x) != S) subduct = true;
    gl_FragColor = n;

// Si el moviment es cap al oest
} else if (move(e.x) == W) {
    if (move(gl_FragColor.x) != W) subduct = true;
    gl_FragColor = e;

// Si el moviment es cap al nord
} else if (move(s.x) == N) {
    if (move(gl_FragColor.x) != N) subduct = true;
    gl_FragColor = s;

// Si el moviment es cap al est
} else if (move(w.x) == E) {
    if (move(gl_FragColor.x) != E) subduct = true;
    gl_FragColor = w;
```

En executar el programa, veurem com es formen aleatòriament i es comencen a expandir les plaques. Si mostrem la dimensió *x* per pantalla, veurem el codi de colors de cada grup de píxels que formen les plaques.

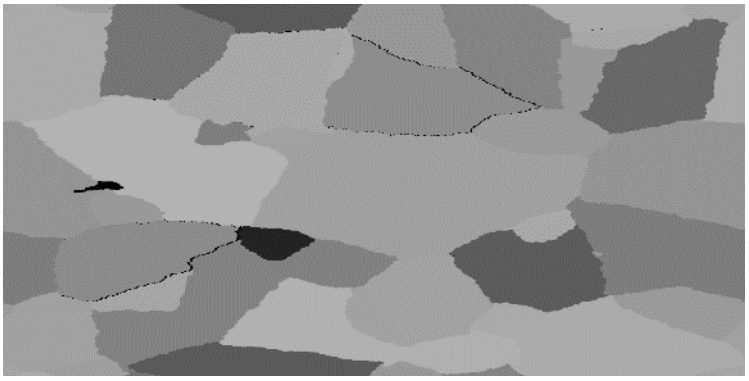

Figura 52 Mapa de plaques generades. Font: pròpia

Les col·lisions de plaques es produeixen quan els píxels de límit d'una placa es mouen cap a una ubicació ocupada anteriorment per píxels pertanyents a una altra placa. Això provoca la subducció, que es modela simplement augmentant lleugerament l'elevació del terreny als llocs de la col·lisió.

```
if (subduct) {
    gl_FragColor.y = 1.;
} else if (uTime < TECTONICS_END_TIME) {
    gl_FragColor.y = max(gl_FragColor.y - 0.0001, 0.);
}
```

Tot i que això només es produeix als píxels al llarg del límit d'una placa, l'impacte s'estén gradualment als píxels veïns mitjançant un model d'erosió tèrmica senzill, que empeny l'elevació d'un píxel en la direcció de la mitjana dels seus veïns.

```
float dz = 0.;
if (abs(e.z - gl_FragColor.z) > 1.) dz += e.z - gl_FragColor.z;
if (abs(w.z - gl_FragColor.z) > 1.) dz += w.z - gl_FragColor.z;
if (abs(n.z - gl_FragColor.z) > 1.) dz += n.z - gl_FragColor.z;
if (abs(s.z - gl_FragColor.z) > 1.) dz += s.z - gl_FragColor.z;
gl_FragColor.z = max(0., gl_FragColor.z + 0.02 * dz);
```

Amb tot això, si mostrem la dimensió *y* per pantalla, veurem l'efecte que generen la formació, moviment i interacció entre plaques.

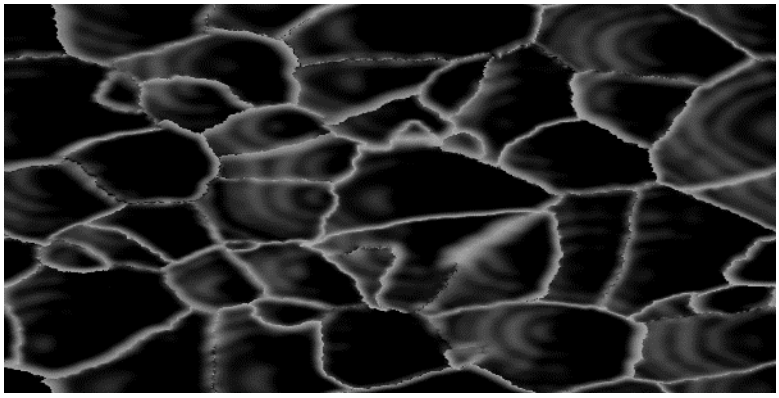

Figura 53 Visualització de les plaques generades en el mapa de subducció. Font: pròpia

En conjunt, podem veure que totes aquestes petites aportacions, proporcionen una simulació decent de la formació de continents amb serralades i en general un terreny accidentat molt interessant.

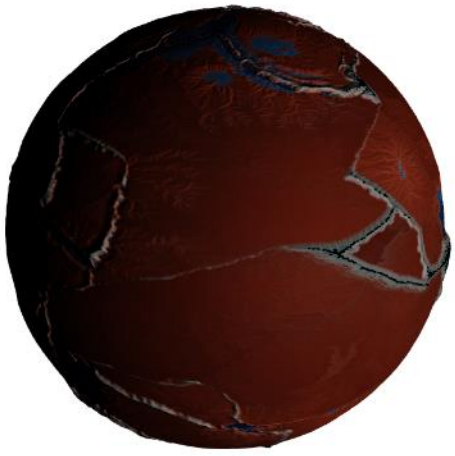

Figura 54 Model tectònic en funcionament. Font: pròpia

5.2.1 MODEL D'AGREGACIÓ

El model d'agregació utilitzat és similar al d'una agregació limitada per difusió, però sense la difusió.

Aquest procés descriu, entre altres coses, la difusió i l'agregació d'ions de zinc en una solució electrolítica sobre elèctrodes. "Difusió" perquè les partícules que formen l'estructura deambulen aleatòriament abans d'unir-se ("agregar") a l'estructura[36].

Exemples d'aquest fenomen es poden trobar en el creixement del corall, el camí que prenen els llamps, la fusió de partícules de pols o de fum i el creixement d'alguns cristalls. Potser el primer estudi seriós d'aquests processos el va fer Witten, T.A. i Sander, L. M. i publicat per ells l'any 1981, titulat: "*Diffusion limited aggregation, a kinetic critical phenomena*" [36].

Hi ha diverses maneres de simular aquest procés per ordinador, en el següent exemple, es parteix d'un punt inicial i s'introdueixen nous punts als costats que caminen aleatòriament fins que estiguin prou a prop per enganxar-se a un píxel existent.

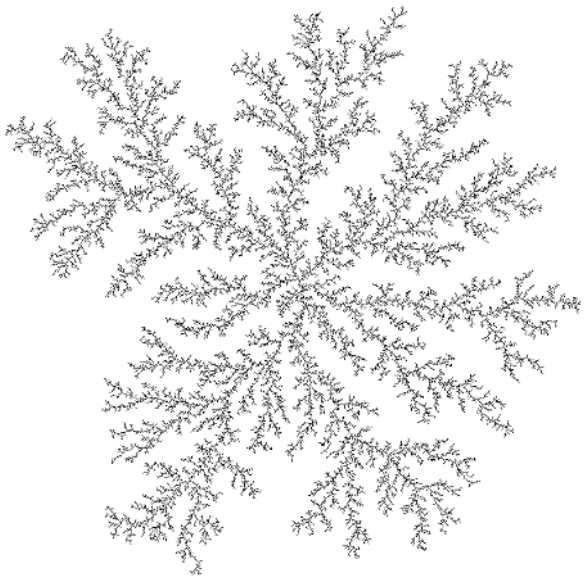

Figura 55 Exemple d'un model d'agregació limitada per difusió amb un punt central inicial. Font: [36]

5.2.2 VISUALITZACIÓ DE LES PLAQUES

De forma similar al que hem fet amb els rius, podem visualitzar de forma més clara el funcionament del nostre model tectònic si representem cada una d'aquestes plaques amb un color distintiu. Per això fem ús d'aquesta funció amb la qual fem ús de tres mapes diferents per acabar mostrant una barreja d'aquests. Fem ús del mapa de plaques de la dimensió *x* , el mapa de subducció de la dimensió *y*, i també el propi mapa del terreny amb la roca i oceans.

```
vec4 map_plates(vec2 fragCoord) {
    vec2 p = fragCoord;
    float q = buf(p).x;
    float uplift = buf(p).y;
    vec4 r = vec4(0,0,0,1);
    r.rgb = (q < 0.) ? vec3(1) : .6 + .6 * cos(2.*PI*q + vec3(0,23,21));
    vec4 fragColor = map_land(fragCoord, false);
    fragColor = r * (5. * fragColor + 3. * clamp(2. * uplift - 1., 0., 1.) + 0.05);
    return fragColor;
}
```

En executar la simulació i arribar al temps establert, comencen a aparèixer les plaques tectòniques que van omplint el planeta. El resultat és el següent:

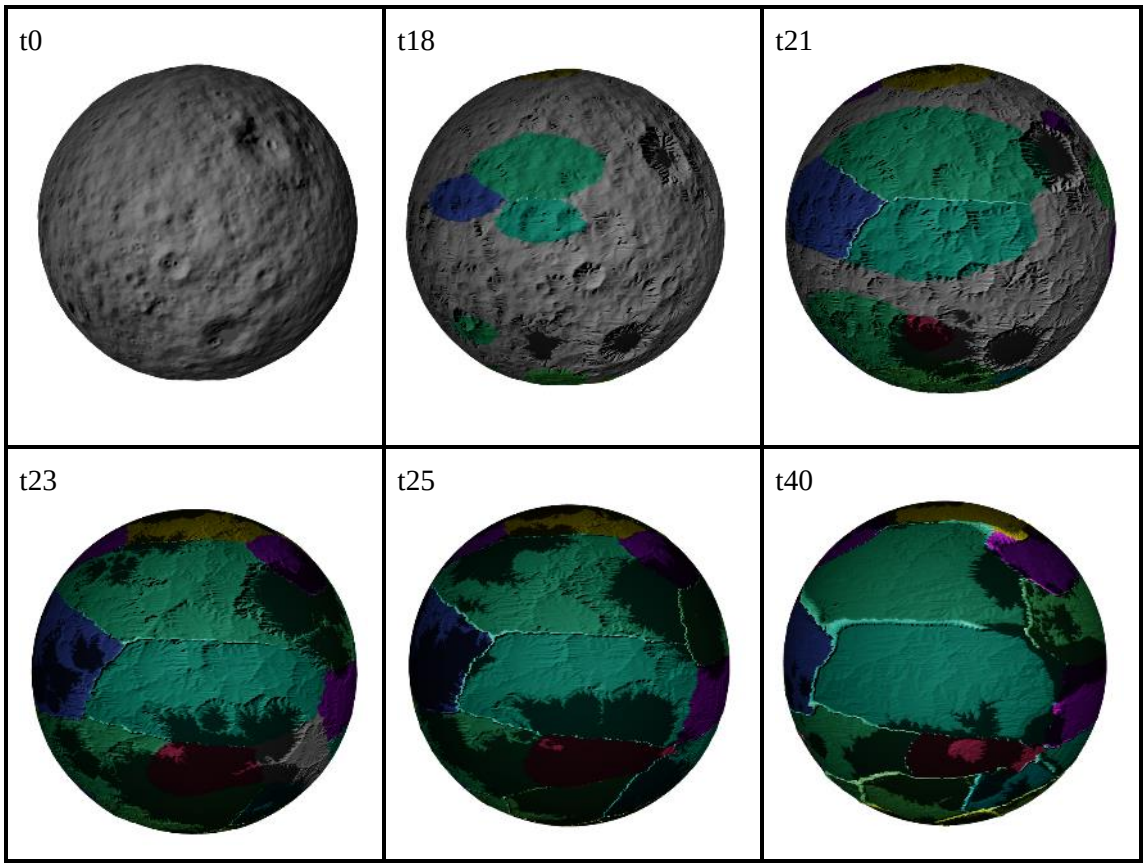

Figura 56 Formació i interacció del model tectònic en executar el programa. Font: pròpia

CAPÍTOL 6. ATMOSFERA

En aquest capítol explorarem alguns aspectes de les atmosferes planetàries, veurem quan i com s'originen i les seves característiques principals. Finalment, implementarem una aproximació realista d'aquest element per integrar-lo a la nostra escena.

6.1 COMPOSICIÓ I FORMACIÓ

L'atmosfera és una barreja de gasos que envolta un planeta. A la Terra, l'atmosfera ajuda a fer possible la vida, ens protegeix dels efectes nocius de la llum solar, la radiació ultraviolada, el vent solar i els raigs còsmics per protegir els organismes del dany genètic. escalfa la superfície del nostre planeta a través de l'efecte hivernacle evita diferències extremes entre les temperatures diürnes i nocturnes. [14]

A més de la Terra, molts dels altres objectes astronòmics del Sistema Solar tenen atmosferes. Aquests inclouen tots els gegants gasosos, així com Mart, Venus, Titià i Plutó. Diverses llunes i altres cossos també tenen atmosferes, com els cometes, el Sol i planetes extrasolars . [18]

La gravetat superficial difereix significativament entre els planetes. Per exemple, la gran força gravitatòria de Júpiter reté gasos lleugers com l'hidrogen i l'heli que s'escapen dels objectes amb menor gravetat. En segon lloc, la distància al Sol determina l'energia disponible per escalfar el gas atmosfèric fins al punt en què alguna fracció del moviment tèrmic de les seves molècules supera la velocitat d'escapament del planeta, permetent-los escapar de la presa gravitatòria d'un planeta. Així, Titià, Tritó i Plutó, distants i freds, són capaços de retenir les seves atmosferes malgrat la seva gravetat relativament baixa.

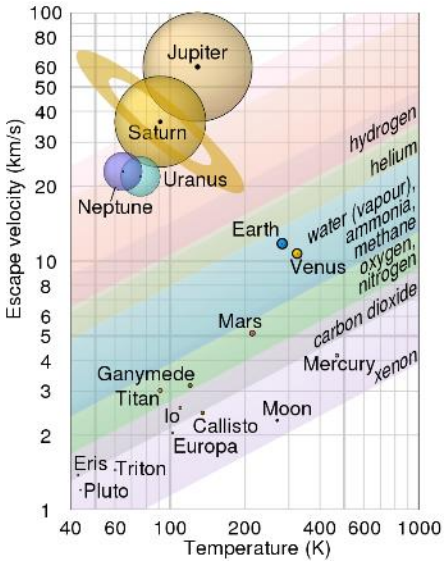

Figura 57 Taula de la velocitat d'escapament en funció de la temperatura superficial d'alguns objectes del Sistema Solar que mostren quins gasos es retenen.. Font: [16]

Atès que una col·lecció de molècules de gas pot estar movent-se a una àmplia gamma de velocitats, sempre n'hi haurà prou ràpid per produir una fuga lenta de gas a l'espai. Les molècules més lleugeres es mouen més ràpidament que les més pesades amb la mateixa energia cinètica tèrmica, de manera que els gasos de baix pes molecular es perden més ràpidament que els d'alt pes molecular. Es creu que Venus i Mart podrien haver perdut gran part de la seva aigua quan, després de ser fotodissociats en hidrogen i oxigen per la radiació ultraviolada solar, l'hidrogen va escapar. El camp magnètic terrestre ajuda a prevenir-ho, ja que, normalment, el vent solar milloraria molt l'escapada d'hidrogen. [18]

Els mecanismes més coneguts que poden causar l'esgotament de l'atmosfera són el vent solar, l'erosió d'impactes i la meteorització.[47]

Les atmosferes tenen efectes espectaculars sobre les superfícies dels cossos rocosos. Els objectes que no tenen atmosfera, o que només tenen una exosfera, tenen un terreny cobert de cràters. Sense atmosfera, el planeta no té protecció contra els meteorits, i tots xoquen amb la superfície com a

meteorits i creen cràters.

La majoria dels meteoroides es cremen com a meteors abans de colpejar la superfície d'un planeta. Quan els meteoroides impacten, els efectes sovint s'esborren per l'acció del vent.[19]

L'erosió eòlica és un factor important en la configuració del terreny dels planetes rocosos amb atmosferes, i amb el temps pot esborrar els efectes tant dels cràters com dels volcans. A més, com que els líquids no poden existir sense pressió, una atmosfera permet que el líquid estigui present a la superfície, donant lloc a llacs, rius i oceans. Se sap que la Terra i Tità tenen líquids a la seva superfície i el terreny del planeta suggereix que Mart tenia líquid a la seva superfície en el passat.

Encara que altres planetes del nostre sistema solar també tenen atmosfera, cap d'ells té la mateixa proporció de gasos i estructura en capes que l'atmosfera terrestre. Actualment, l'atmosfera de la Terra està formada per nitrogen (78%), oxigen (21%), argó (0,9%), diòxid de carboni (0,04%) i gasos traça. Però no sempre ha sigut així, aquesta composició és el producte de milers de milions d'anys de modificació bioquímica per part dels organismes vius. [15]

Just després de la seva creació, es creu que la Terra tenia una atmosfera fina composta principalment per gasos d'heli (He) i hidrogen (H). La gravetat de la Terra no va poder contenir aquests gasos lleugers i van escapar fàcilment a l'espai exterior [2]

Durant els següents centenars de milions d'anys, a mesura que el planeta va canviar i es va començar a formar l'escorça, es van produir erupcions volcàniques amb freqüència. Aquests volcans van bombejar vapor d'aigua, amoníac i diòxid de carboni a l'atmosfera al voltant de la Terra. Això va començar a crear una atmosfera més gruixuda composta per una gran varietat de gasos. [2]

Després de refredar-se, la Terra jove tenia una atmosfera amb el diòxid de carboni com a constituent principal, així com nitrogen i una mica d'aigua. La pressió superficial també era molt més alta, gairebé cent vegades la d'avui i l'atmosfera era molt més alta, a causa de la superfície calenta. Aquestes característiques la feien més semblant a l'atmosfera de Venus actual que a la de la Terra actual.[54]

Figura 58 Una il·lustració artística de la Terra actual i fa 4.500 milions d'anys. Font: [16]

6.2 IMPLEMENTACIÓ D'UN MODEL ATMOSFÈRIC

Potser us heu preguntat per què el cel sol ser blau, però vermell al capvespre. El fenomen òptic que és (principalment) responsable d'això s'anomena dispersió de Rayleigh.

Moltes generacions de físics i matemàtics abans del segle XIX-XX també es van obsessionar amb intentar esbrinar què podria fer que el cel aparegués taronja a la posta i la sortida del sol i blau durant el dia. Històricament, el descobriment de la dispersió atmosfèrica s'atribueix a lord Rayleigh (John William Strutt), un físic anglès premi Nobel de Física (1842-1919).[20]

Generar una dispersió atmosfèrica realista en ordinador sempre ha estat un problema difícil, però és molt important per renderitzar entorns exteriors realistes. Les equacions que descriuen la dispersió atmosfèrica són tan complexes que s'han dedicat llibres sencers al tema. Els models de gràfics per ordinador generalment utilitzen equacions simplificades, i molt pocs funcionen a velocitats de fotogrames interactives.[21]

6.2.1 DISPERSIÓ ÚNICA

En una habitació sense llum, esperaries no veure res. Els objectes només es fan visibles quan un raig de llum rebota en ells i colpeja els nostres ulls. La llum sempre viatja a través d'un mitjà. En el nostre cas, aquest medi és l'aire que respirem. Com a resultat, l'aspecte dels objectes es veu afectat per la quantitat d'aire que travessa. A la superfície de la Terra, la densitat de l'aire és relativament baixa; la seva contribució és tan petita que només es pot apreciar realment quan la llum recorre grans distàncies. Les muntanyes llunyanes es barregen amb el cel, tot i que els objectes propers a nosaltres semblen que pràcticament no els afecta la dispersió atmosfèrica.

El primer pas per replicar els efectes òptics de la dispersió atmosfèrica és entendre com la llum viatja a través d'un medi com l'aire. Com hem dit abans, només podem veure alguna cosa quan la llum arriba als nostres ulls. En el context dels gràfics en 3D, el nostre ull és la càmera que s'utilitza per renderitzar l'escena. Les molècules que formen l'aire que ens envolta poden desviar els raigs de llum que les travessen. Per tant, tenen el poder d'alterar la manera com percebem els objectes. Vist de forma molt simplificada, hi ha dues maneres en què les molècules de l'aire poden afectar la nostra visió. [22]

6.2.2 DISPERSIÓ EXTERIOR (OUT-SCATTERING)

La forma més òbvia en què les molècules d'aire interactuen amb la llum és desviant-la, canviant-ne la direcció. Si un raig de llum dirigit a colpejar la càmera es desvia, estem davant d'un procés anomenat "dispersió exterior" o "out-Scattering".

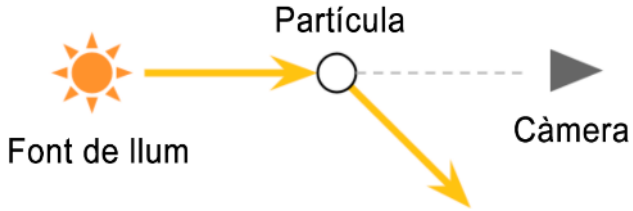

Figura 59 Dispersió out-scattering. Font: [22]

Una font de llum real pot emetre quadrilions (10^{24}) de fotons cada segon, i cadascun té una certa probabilitat de colpejar una molècula d'aire. Com més dens sigui el medi en què viatja la llum, més probabilitats hi ha que un sol fotó es desvii. La intensitat amb què la dispersió afecta la llum també depèn de la distància recorreguda.

La dispersió exterior fa que la llum es torni progressivament més tènue i depèn tant de la distància recorreguda com de la densitat de l'aire.[22]

6.2.3 DISPERSIÓ INTERNA (IN-SCATTERING)

Quan la llum és desviada per una partícula, també pot passar que es redirigeixi cap a la càmera. Això és efectivament el contrari de la dispersió exterior (out-scattering) i, s'anomena dispersió interna (in-scattering).

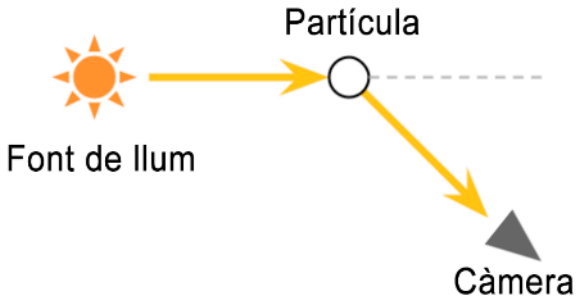

Figura 60 Dispersió interna (in-scattering). Font: [22]

En determinades condicions, la dispersió interna permet veure fonts de llum que no es troben a la llum directa de la càmera. El seu efecte òptic més evident dona lloc a halos de llum al voltant de fonts de llum. Són causats pel fet que la càmera rep raigs de llum directes i indirectes de la mateixa font, amplificant el nombre de fotons rebuts. [22]

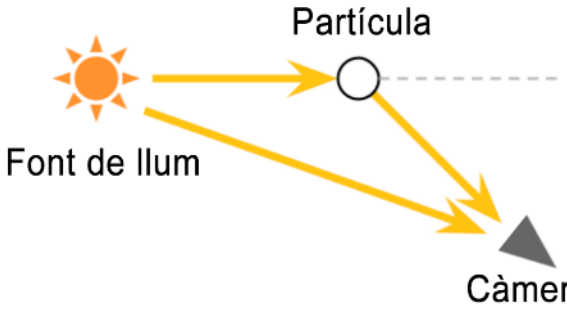

Figura 61 Dispersió interna (in-scattering) amb font de llum directa. Font: [22]

6.2.4 DISPERSIÓ VOLUMÈTRICA ÚNICA

Un sol raig de llum es pot desviar un nombre arbitrari de vegades. Això vol dir que la llum pot recórrer camins molt complexos abans d'arribar a la càmera. Això suposa un repte important, ja que renderitzar materials translúcids amb alta fidelitat requereix simular el recorregut de cada raig de llum de forma individual. Això s'anomena Traçat de raigs (*raytracing*) i encara que avui dia és possible computacionalment aconseguir una representació en temps real, requereix de hardware d'alta gamma. La tècnica que veurem es coneix com a dispersió única, ja que només té en compte un únic esdeveniment de dispersió per a un raig de llum. Més endavant veurem com aquesta simplificació encara permet obtenir resultats realistes a una fracció del cost que tindria el raytracing real.

La clau per crear un resultat realista és simular què passa amb els raigs de llum quan viatgen per l'atmosfera d'un planeta. El diagrama següent mostra una càmera mirant a través d'un planeta. La idea bàsica darrere d'aquesta tècnica de renderització és calcular com la llum que viatja d'A a B es veu afectada per la dispersió. Això significa calcular les contribucions que tenen la dispersió exterior i interna sobre la llum que viatja cap a la càmera. Com s'ha comentat abans, experimentem una atenuació a causa de la dispersió exterior. La quantitat de llum present a cada punt $P \in \underline{AB}$ té una petita probabilitat de ser desviada lluny de la càmera.[22]

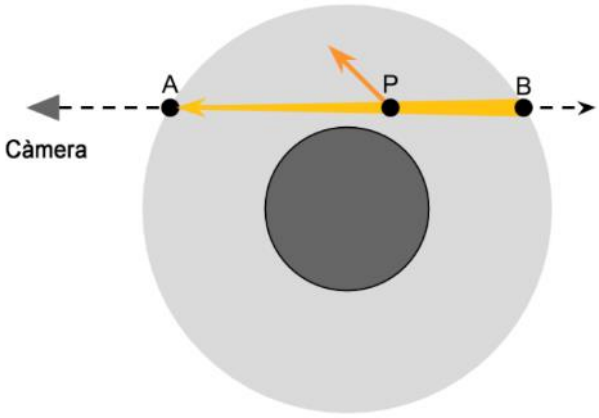

Figura 62 Diagrama de la interacció de la llum amb l'atmosfera amb desviació. Font: [22]

Per tenir en compte correctament quanta dispersió es produeix a cada punt P, primer hem de saber quanta llum hi havia a P en primer lloc. Suposant que només hi ha una sola estrella que il·lumina el planeta, tota la llum que rep P ha de provenir del sol. Part d'aquesta llum estarà sotmesa a una dispersió interna i es desviarà accidentalment cap a la càmera.

Aquests dos passos són suficients per aproximar la majoria dels efectes que es poden observar a l'atmosfera. No obstant això, les coses es compilen pel fet que la quantitat de llum que rep P del sol està subjecta a la dispersió mentre viatja per l'atmosfera de C a P. [22]

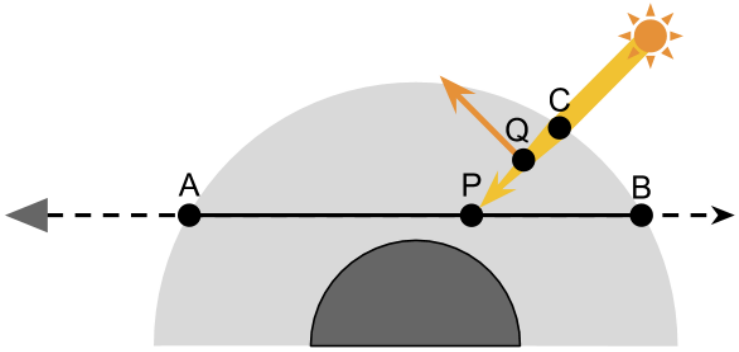

Figura 63 Diagrama de la interacció de la llum amb l'atmosfera amb dispersió. Font: [22]

En resumides comptes:

- La línia de visió de la càmera entra a l'atmosfera en A, i surt en B.
- La quantitat de llum que rep P prové del sol.
- La quantitat de llum que rep P està sotmesa a dispersió, ja que viatja per l'atmosfera $\underline{AB}$.
- Una part de la llum rebuda per P està sotmesa a una dispersió interna, que envia en direcció a la càmera.
- Una part de la llum de P que es dirigeix cap a la càmera està sotmesa a dispersió i es desvia lluny de la línia de visió.
- Com a aproximació, tenim en compte les contribucions de la dispersió externa i interna a mesura que passa per cada punt $P \in \underline{AB}$.

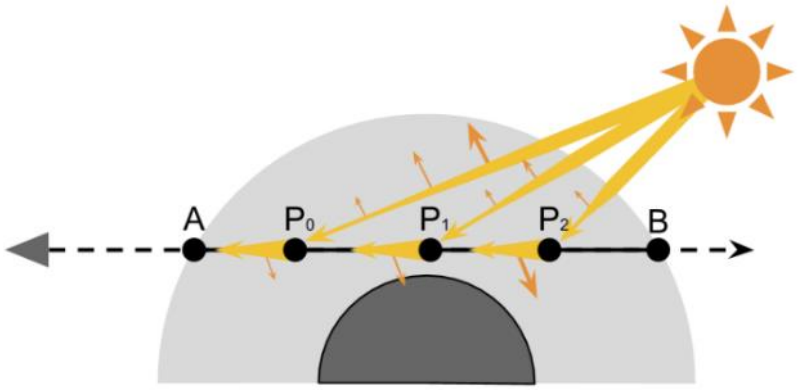

Figura 64 Diagrama de les contribucions de la llum en travessar l'atmosfera. Font: [22]

Aquesta no és l'única manera que els raigs de llum poden arribar a la càmera. La solució que es proposa té en compte la dispersió interna al llarg de la línia de visió $\underline{AB}$. La llum que arriba a un punt P des del sol té una certa probabilitat de ser desviada cap a la càmera.

Tanmateix, hi ha molts més camins que els raigs de llum poden prendre per arribar a la càmera. Per exemple, un dels raigs dispersos en el seu camí cap a P es pot dispersar cap a la càmera en una segona col·lisió (diagrama a continuació). I podria haver-hi raigs que arribin a la càmera després de ser desviats tres, quatre o cinc vegades.

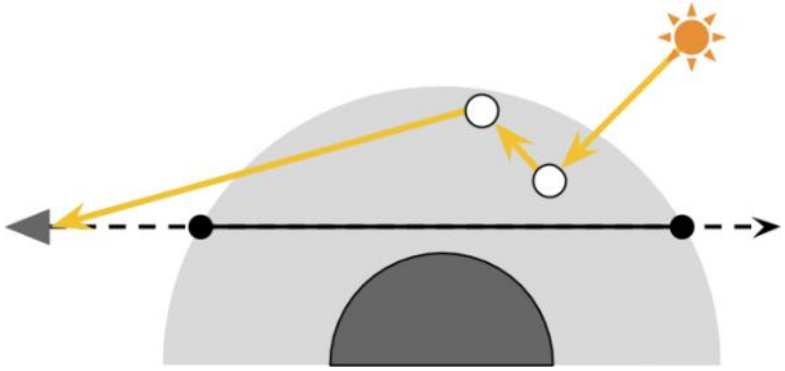

Figura 65 Diagrama dels diferents camins que pot recorre la llum fins a arribar a la càmera. Font: [22]

Aquesta tècnica s'anomena dispersió única perquè només té en compte la dispersió interna que es produeix al llarg de la línia de visió. Les tècniques més avançades poden estendre aquest procés tenint en compte altres maneres en què la llum pot arribar a la càmera. El nombre de camins possibles, però, creix exponencialment amb el nombre d'esdeveniments de dispersió que es tenen en compte. Per sort, la probabilitat d'arribar a la càmera també disminueix de manera exponencial. [22]

6.2.5 SHADER INICIAL

El primer pas serà escriure un shader de postprocessament que s'encarregarà de dibuixar l'atmosfera al voltant del nostre planeta. Aquest programa s'executarà per a cada píxel de la pantalla i li donarà un color segons els nostres càlculs.

El primer que haurem d'esbrinar es on es creua la vista de la càmera amb una esfera al voltant. Obtindrem els punts d'origen i direcció del raig, així com el punt on intersecta amb l'atmosfera, tant el de entrada com el de sortida. En cas de no arribar a sortir per l'atmosfera i impactar amb el planeta també es tindrà en compte.

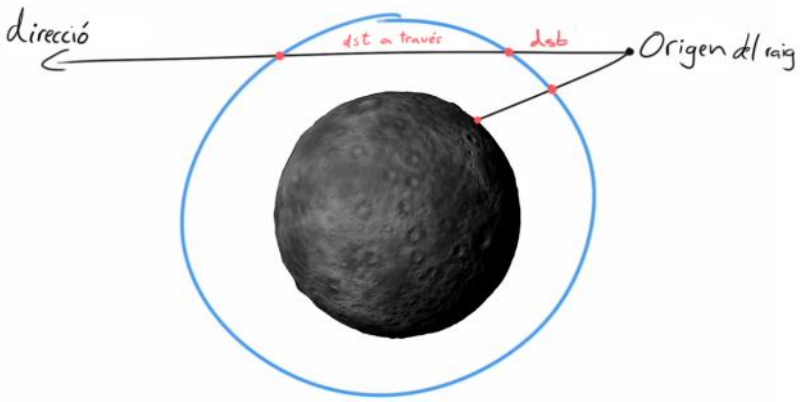

Figura 66 Concepte del raig de llum que creua l'atmosfera del planeta. Font: pròpia

Aquest fragment de codi ens servirà per comprovar aquest funcionament bàsic de forma que dibuixarem a cada píxel la quantitat d'atmosfera que ha de travessar el raig.

```
void main() {
        // Calcula les coordenades globals relatives a la pantalla
        float z = texture2D(tDepth, vUv).x;
        vec3 posWorldScreen = _ScreenToWorld(vec3(vUv, z));

        // Calcula origen i direcció del raig
        vec3 rayOrigin = cameraPosition;
        vec3 rayDirection = normalize(posWorldScreen - cameraPosition);

        // Dispara un raig al planeta i calcula la distància
        float dstToSurface = raySphere(planetPosition, planetRadius, rayOrigin,
rayDirection).x;

        // Dispara un raig a l'atmosfera i calcula la distància
        vec2 hitInfo = raySphere(planetPosition, atmosphereRadius, rayOrigin,
rayDirection);
        float dstToAtmosphere = hitInfo.x;
        float dstThroughAtmosphere = min(hitInfo.y, dstToSurface -
dstToAtmosphere);

        // Dibuixa el resultat
        vec3 atmos = vec3(dstThroughAtmosphere / (atmosphereRadius * 2.0));
        gl_FragColor = vec4(atmos, 0.0);
}
```

En la següent imatge podem veure el resultat, el raig que no impacta amb l'atmosfera no afecta el color final, però a mesura que va apropant-se al centre i travessa més quantitat d'atmosfera, té un

color blanc més intens fins a arribar a impactar amb el terreny i tallant així la distància notablement.

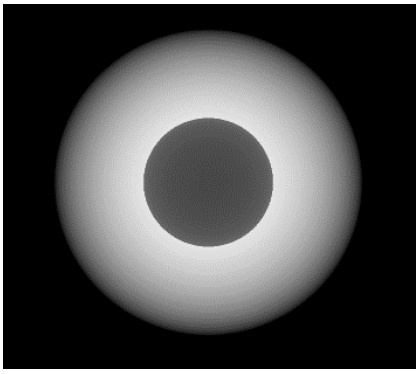

Figura 67 Primera prova del shader atmosfèric. Font: pròpia

Aquest resultat és un bon punt de partida, però per poder obtenir un resultat més proper a la realitat, hem de conèixer en profunditat com interactua la llum amb els elements que formen l'atmosfera.

6.2.6 FUNCIÓ DE TRANSMISSIÓ

Per calcular la quantitat de llum transmesa a la càmera, és útil fer el mateix viatge que experimenten els raigs de llum del sol. Mirant el diagrama següent, és fàcil veure que els raigs de llum que arriben a C viatgen per l'espai buit. Com que no hi ha res al seu pas, tota la llum arriba a C sense estar sotmesa a cap efecte de dispersió. Indiquem amb I_C la quantitat de llum no dispersa que rep el punt C del sol. Durant el seu viatge cap a P, la llum entra a l'atmosfera del planeta. Alguns raigs xocaran amb les molècules suspeses a l'aire, rebotant en diferents direccions. El resultat és que part de la llum s'allunya del camí. La quantitat de llum que arriba a P (indicada amb I_P) serà inferior a I_C candeles (cd).[20][21][22]

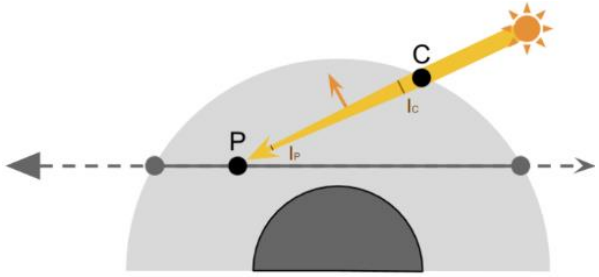

Figura 68 Diagrama de la funció de transmissió. Font: [22]

La relació entre I_C i I_P s'anomena **transmitància**:

$$T(\underline{CP}) = \frac{I_P}{I_C}$$

Fórmula (6)

I el podem utilitzar per indicar el percentatge de llum que no es dispersa durant el trajecte de C a P.

Més endavant explorarem la naturalesa d'aquesta funció. De moment, l'únic que cal saber és que correspon a l'aportació de la dispersió atmosfèrica.

En conseqüència, la quantitat de llum que rep P és:

$$\boldsymbol{I_P = I_C\, T(\overline{\mathrm{CP}})}$$

Fórmula (7)

6.2.7 FUNCIÓ DE DISPERSIÓ

El punt P rep la llum directament del sol. Tanmateix, no tota la llum que passa per P es torna a transferir a la càmera. Per calcular quanta llum arriba realment a la càmera, hem d'introduir un altre concepte: la funció de dispersió S. La seva finalitat és indicar quanta llum es desvia en una determinada direcció. Si mirem el diagrama següent, ens adonem que només aquells raigs de llum que es desvien per un angle de θ es dirigeixen cap a la càmera. [22]

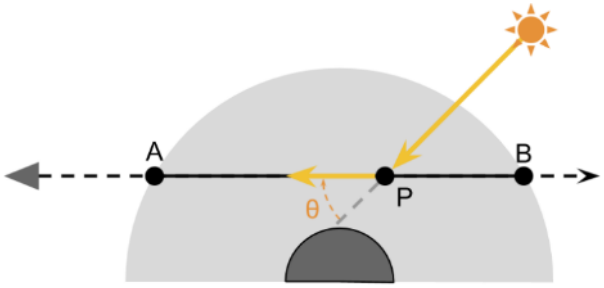

Figura 69 Diagrama de la funció de dispersió. Font: [22]

El valor de $S(\lambda, \theta, h)$ indica la proporció de llum desviada per θ radians. Aquesta funció és el centre del nostre problema. De moment, l'únic que cal saber és que depèn del color de la llum entrant (indicat per la seva longitud d'ona λ), de l'angle de dispersió θ i de l'altitud h de P. El motiu pel qual l'altitud és important és perquè la densitat atmosfèrica canvia en funció de l'altitud. I la densitat és, en definitiva, un dels factors per determinar quanta llum es dispersa. [22]

Ara tenim totes les eines necessàries per escriure una equació general que mostri quanta llum es transfereix de P a A:

$$I_{PA} = I_P\ \mathrm{S}(\lambda, \theta, h)\ \mathrm{T}\ (\overline{\mathrm{PA}})$$

Fórmula (8)

Podem expandir I_P utilitzant la definició anterior:

$$I_{PA} = I_C\, T(\overline{\mathrm{CP}})\ \mathrm{S}(\lambda, \theta, h)\ \mathrm{T}\ (\overline{\mathrm{PA}}) =$$

$$= \underbrace{I_C\, S(\lambda, \theta, h)}_{\text{in-scattering}}\ \underbrace{T(\overline{\mathrm{CP}})\, T(\overline{\mathrm{PA}})}_{\text{out-scattering}}$$

Fórmula (9)

- La llum viatja del sol a C, sense dispersar-se en el buit de l'espai;
- La llum entra a l'atmosfera i viatja de C a P. En fer-ho, només la fracció $T(\overline{\mathrm{CP}})$ arriba al seu destí a causa de la **dispersió externa** (out-scattering);
- Part de la llum que ha arribat a P des del sol es desvia cap a la càmera. La proporció de llum sotmesa a la **dispersió interna** (in-scattering) és $S(\lambda, \theta, h)$;
- La llum restant viatja de P a A, i una vegada més només es transmet la fracció $T(\overline{\mathrm{PA}})$.

6.2.8 INTEGRACIÓ NUMÈRICA

En paràgrafs anteriors hi ha una inconsistència en la manera com s'ha escrit la intensitat. El símbol I_{PA} indica la quantitat de llum transferida de P a A. Tanmateix, aquesta quantitat no té en compte tota la llum que rep A. En aquests models simplificats de dispersió atmosfèrica, estem tenint en compte la dispersió interna des de cada punt de la línia de visió de la càmera a través de l'atmosfera.

La quantitat total de llum que rep A, I_A, es calcula sumant la contribució de tots els punts $P \in \underline{AB}$. Matemàticament parlant, hi ha una infinitat de punts al segment $\underline{AB}$, de manera que serà impossible fer un bucle per tots ells. El que podem fer, però, és dividir $\underline{AB}$ en molts segments més petits de longitud ds (diagrama a continuació) i acumular la contribució de cadascun.

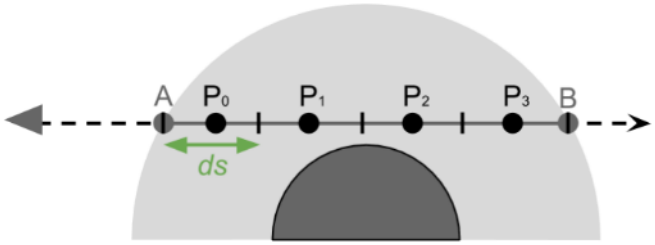

Figura 70 Diagrama de la integració numèrica. Font: [22]

Aquest procés d'aproximació s'anomena integració numèrica i condueix a la següent expressió:

$$I_A = \sum_{P \in \overline{AB}} I_{PA}\, ds$$

Fórmula (10)

Com més punts tinguem en compte, més precís serà el nostre resultat final. En realitat, el que haurem de fer al nostre shader atmosfèric és recórrer diversos punts P_i dins de l'atmosfera, acumulant la seva contribució al resultat global.

6.2.9 LLUM DIRECCIONAL

Si el sol o estrella està relativament a prop, és millor modelar-lo com a font de llum puntual. En aquest cas, la quantitat de llum rebuda a C depèn de la distància del sol. Si parlem de planetes, sovint podem suposar que el sol està tan lluny que els seus raigs arriben al planeta tots amb el mateix angle. Si aquest és el cas, podem modelar el sol com a font de llum direccional. La llum rebuda de fonts de llum direccionals es manté constant independentment de la distància que recorre.

Per tant, cada punt C rep la mateixa quantitat de llum i la direcció cap al sol és la mateixa per a tots els punts.

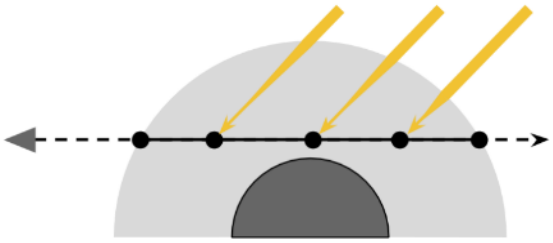

Figura 71 Simplificació amb llum direccional. Font: [22]

Podem utilitzar aquesta hipòtesi per simplificar el nostre conjunt d'equacions.

Substituïm I_C per la constant I_S, que representa la intensitat del sol o estrella.

$$I_A = \sum_{P \in \underline{AB}} I_{PA}\, ds \;\; =$$

$$= \sum_{P \in \overline{AB}} I_C\, S(\lambda, \theta, h)\, T(\overline{CP})\, T(\overline{PA})\, ds =$$

$$= I_S \sum_{P \in \overline{AB}} S(\lambda, \theta, h)\, T(\overline{CP})\, T(\overline{PA})\, ds$$

Fórmula (11)

Hi ha una altra optimització que podem realitzar, i implica la funció de dispersió $S(\lambda, \theta, h)$. Si la llum solar prové sempre de la mateixa direcció, l'angle θ esdevé una constant. Veurem com la contribució direccional de $S(\lambda, \theta, h)$ es pot descomptar de la suma. De moment, deixem-ho dins.

6.2.10 DISPERSIÓ DE RAYLEIGH

La interacció entre la llum i la matèria és extremadament complexa i no hi ha una manera fàcil de descriure-la completament. Modelar la dispersió atmosfèrica és, de fet, excepcionalment difícil. Part del problema prové del fet que l'atmosfera no és un medi homogeni. Tant la seva densitat com la seva composició canvien significativament en funció de l'altitud, fent pràcticament impossible trobar un model "perfecte".

És per això que la literatura científica presenta diversos models de dispersió, cadascun dissenyat per descriure un subconjunt de fenòmens òptics que ocorren en condicions específiques. La majoria dels efectes òptics que presenten els planetes es poden reproduir tenint en compte dos models diferents: la dispersió Rayleigh i la dispersió Mie. Aquestes dues eines matemàtiques permeten predir com es dispersa la llum sobre objectes de diferent mida. Els primers modelen com la llum es reflecteix per les molècules d'oxigen i nitrogen que formen la major part de l'aire. Aquest últim modela com la llum es reflecteix en compostos molt més grans que es troben en suspensió a l'atmosfera baixa, com el pol·len, la pols i els contaminants.[48]

La dispersió de Rayleigh fa que el cel sigui blau i les postes de sol vermelles. La dispersió de Mie dona als núvols el seu color blanc.[49]

Imaginem un raig de llum viatjant per l'espai buit, xocant de sobte amb una partícula. El resultat d'aquesta col·lisió varia dràsticament depenent de la mida de la partícula i del color de la llum. Si la partícula és prou petita (com àtoms i molècules), el comportament de la llum es prediu millor mitjançant la dispersió de Rayleigh.

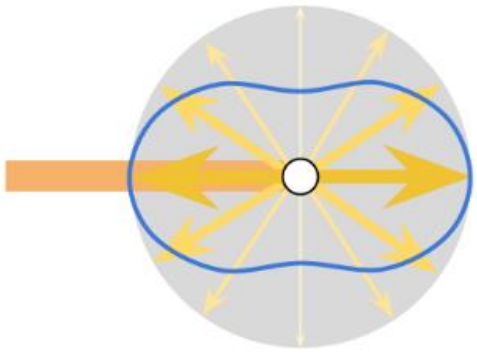

Figura 72 Possibles direccions de desviació d'un fotó en interactuar amb una partícula. Font: [22]

El que passa és que una part de la llum continua el seu viatge sense ser afectada. Tanmateix, un petit percentatge d'aquesta llum original interactua amb la partícula i es dispersa en totes direccions. No totes les direccions, però, reben la mateixa quantitat de llum. És més probable que els fotons passin directament a través de la partícula o que rebotin. Per contra, el resultat menys probable per a un fotó és desviar-se 90 graus. Aquest comportament es pot veure al diagrama següent. La línia blava mostra les direccions preferides per a la llum dispersa.[22]

Aquest fenomen òptic es descriu matemàticament per l'equació de dispersió de Rayleigh $S(\lambda, \theta, h)$, que indica la proporció de la llum original I_0 que es dispersa cap a la direcció θ:

$$I = I_0 \, S(\lambda, \theta, h)$$

$$S(\lambda, \theta, h) = \frac{\pi^2 \left(n^2 - 1\right)^2}{2} \underbrace{\frac{\rho(h)}{N}}_{\text{density}} \overbrace{\frac{1}{\lambda^4}}^{\text{wavelength}} \underbrace{\left(1 + \cos^2 \theta\right)}_{\text{geometry}}$$

Fórmula (12)

On:

- λ: la longitud d'ona de la llum entrant (m);
- θ: l'angle de dispersió (graus);
- h: altitud del punt (m);
- n=1,00029: l'índex de refracció de l'aire;
- N=$2{,}504 \cdot 10^{25}$: la densitat del nombre molecular de l'atmosfera estàndard. Aquest és el nombre de molècules per metre cúbic ;
- $\rho(h)$: la relació de densitat. Aquest nombre és igual a 1 al nivell del mar, i disminueix exponencialment amb h.

El primer que hem observat sobre la dispersió de Rayleigh és que determinades direccions reben més llum que altres. El segon aspecte important és que la quantitat de llum dispersa depèn fortament de la longitud d'ona λ de la llum entrant. El diagrama polar següent visualitza $S(\lambda, \theta, h)$ per a tres longituds d'ona diferents. Avaluar S amb h=0 sovint es coneix com a dispersió al nivell del mar. [22]

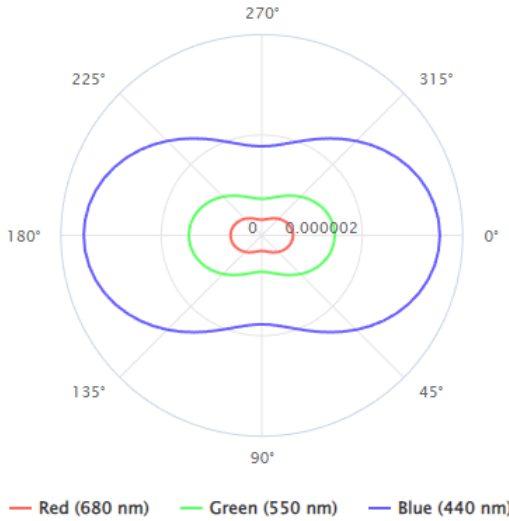

Figura 73 Diagrama polar de dispersió per a tres longituds d'ona (vermell, verd i blau). Font: [22]

La següent imatge mostra una representació dels coeficients de dispersió per a un rang continu de longitud d'ona/color de l'espectre visible

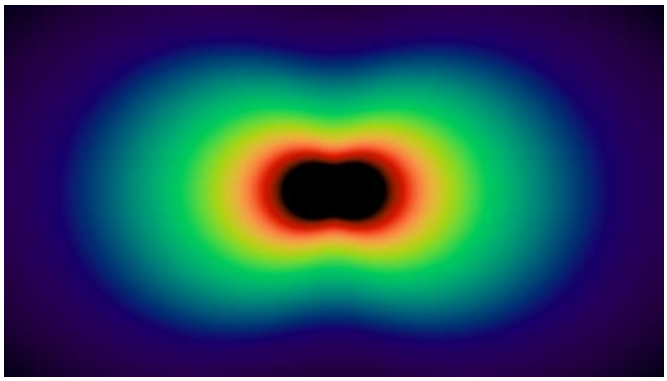

Figura 74 Representació dels coeficients de dispersió. Font: [23]

El centre de la imatge apareix és negre perquè les longituds d'ona d'aquest rang estan fora de l'espectre visible.

6.2.11 COEFICIENT DE DISPERSIÓ DE RAYLEIGH

L'equació per a la dispersió de Rayleigh indica quanta llum es dispersa cap a una direcció determinada. No diu, però, quanta energia es dispersa en total. Per calcular-ho, hem de tenir en compte la dispersió d'energia en totes direccions. Aquesta derivació resulta en:

$$\beta(\lambda, h) = \frac{8\pi^3 \left(n^2 - 1\right)^2}{3} \frac{\rho(h)}{N} \frac{1}{\lambda^4}$$

Fórmula (13)

On $\beta(\lambda, h)$ indica la fracció de l'energia que es perd per dispersió després d'una col·lisió amb una sola partícula. Sovint s'anomena coeficient de dispersió de Rayleigh.[22]

β és el coeficient d'extinció utilitzat en la definició de la transmitància $T(\overline{\mathrm{AB}})$ sobre un segment $\overline{AB}$. Malauradament, calcular β és molt car computacionalment. Al shader que anem a escriure, intentarem estalviar el màxim de càlcul possible. Per aquest motiu, és útil "extreure" del coeficient de dispersió tots els factors que són constants. Això ens deixa amb $\beta(\lambda)$, que es coneix com el coeficient de dispersió de Rayleigh com a nivell del mar (h=0):

$$\beta(\lambda) = \frac{8\pi^3 \left(n^2 - 1\right)^2}{3} \frac{1}{N} \frac{1}{\lambda^4}$$

Fórmula (14)

Aquesta nova equació dona una altra manera d'entendre com es dispersen els diferents colors de la llum. Aquest gràfic mostra la quantitat de llum dispersa a la qual està sotmesa, en funció de la seva longitud d'ona:

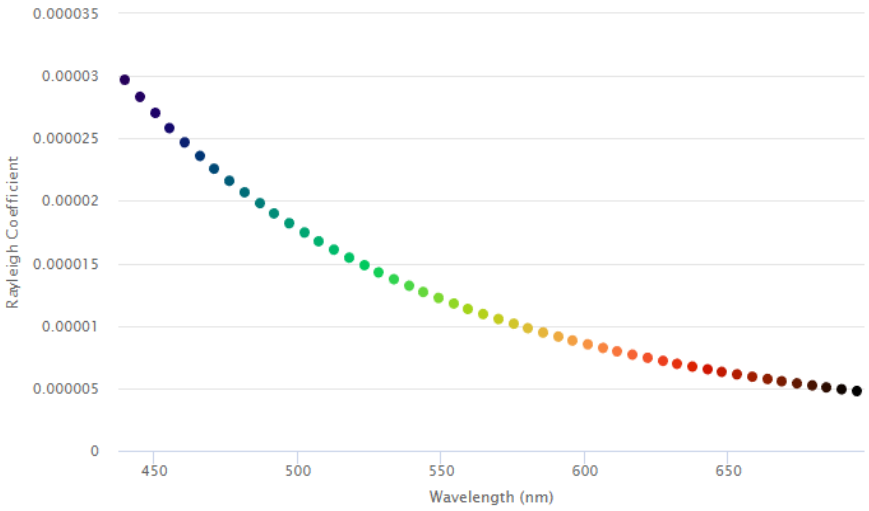

Figura 75 Dispersió de la **llum en funció de la longitud d'ona λ . Font: [22]**

És la forta relació entre el coeficient de dispersió β i λ que torna vermells els cels de la posta de sol. Els fotons que rebem del sol es distribueixen en una àmplia gamma de longituds d'ona. El que ens diu la dispersió de Rayleigh és que els àtoms i les molècules de l'atmosfera terrestre dispersen la llum blava amb més força que la verda o la vermella. Si es permet que la llum viatgi durant prou temps, tota la seva llum blava es perdrà per dispersió. Això és exactament el que passa al capvespre, quan la llum viatja gairebé paral·lela a la superfície. [22]

Amb el mateix raonament, podem entendre per què el cel apareix blau. La llum del sol arriba amb una direcció específica. Tanmateix, el seu component blau està dispers en totes direccions. Quan mirem el cel, la llum blava prové de totes direccions.

6.2.12 RELACIÓ DE DENSITAT ATMOSFÈRICA

Des d'un punt de vista lògic, té sentit que la força de la dispersió sigui proporcional a la densitat atmosfèrica. Més molècules per metre quadrat significa més possibilitats que els fotons es dispersin. El repte és que la composició de l'atmosfera és molt complexa, formada per diverses capes amb diferents pressions, densitats i temperatures. Per sort, la major part de la dispersió de Rayleigh es produeix en els primers quilòmetres de l'atmosfera. [22]

El següent diagrama mostra la relació entre la densitat i l'altitud a la atmosfera terrestre.

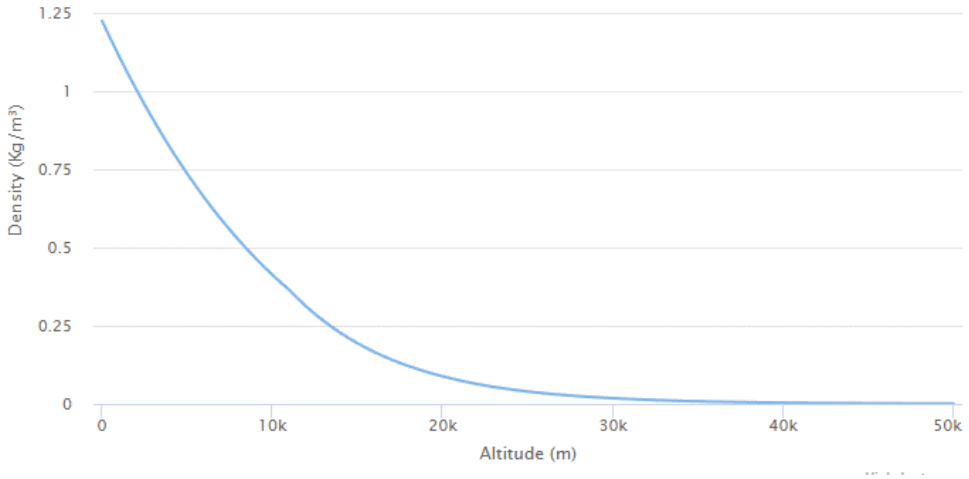

Figura 76 Relació entre la densitat de l'atmosfera terrestre i l'altitud respecte de la superfície de la Terra. Font: [22]

El valor de $\rho(h)$ representa la mesura atmosfèrica a una altitud de h metres, normalitzada de manera que comenci a zero. En molts articles científics, ρ també s'anomena proporció de densitat perquè també es pot definir com:

$$\rho(h) = \frac{densitat(h)}{densitat(0)}$$

Fórmula (15)

Dividint la densitat real per $densitat(0)$ fa que $\rho(h)$ sigui 1 al nivell del mar. Com s'ha destacat abans, però, la densitat de càlcul $densitat(h)$ està lluny de ser trivial. Ho podem aproximar amb una corba exponencial; De fet, probablement t'has adonat que la densitat a l'atmosfera segueix una decadència exponencial.

Si volem aproximar la relació de densitat amb una corba exponencial, ho podem fer així:

$$\rho(h) = exp\left\{-\frac{h}{H}\right\}$$

Fórmula (16)

On H és un factor d'escala anomenat alçada d'escala. Per a la dispersió de Rayleigh a l'atmosfera de la Terra, sovint se suposa que H=8500 metres (diagrama a continuació). Per a la dispersió de Mie, sovint és d'uns 1200 metres.[52]

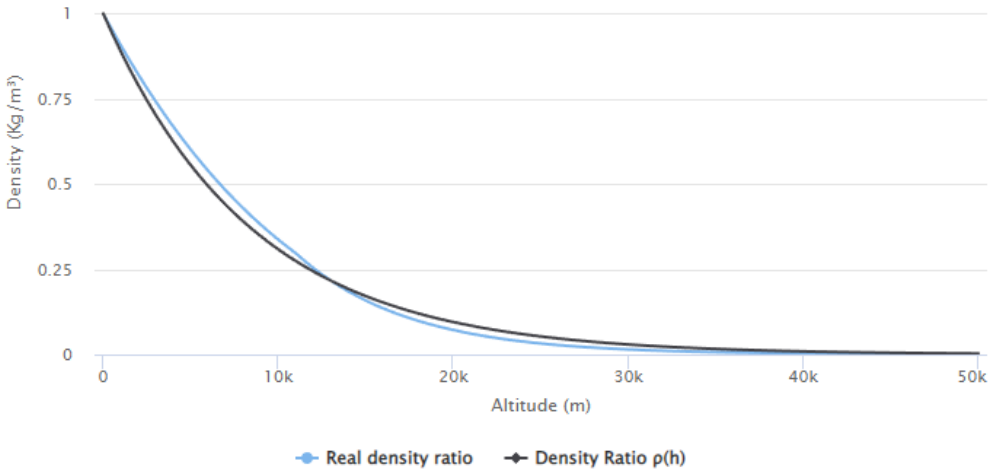

Figura 77 Comparació entre la relació de densitat real i p(h). Font: [22]

El valor utilitzat per a H no dona la millor aproximació per a $\rho(h)$. Això, però, no és realment un problema. La majoria de les quantitats presentades estan sotmeses a aproximacions severes.

6.2.13 DECADÈNCIA EXPONENCIAL

Anteriorment hem vist com tenir en compte la dispersió a la qual està sotmès un raig de llum després d'interaccionar amb una sola partícula. La quantitat utilitzada per modelar aquest fenomen es l'anomenat coeficient de dispersió β. Hem introduït els coeficients β per tenir-ho en compte.

En el cas de la dispersió de Rayleigh, també hem vist una forma per calcular la quantitat de llum que està sotmesa a la dispersió atmosfèrica per interacció única:

$$\beta(\lambda, h) = \frac{8\pi^3 \left(n^2 - 1\right)^2}{3} \frac{\rho(h)}{N} \frac{1}{\lambda^4}$$

Fórmula (17)

Quan avaluem aquesta funció al nivell del mar, és a dir, h=0, produeix els següents resultats:

$$\beta(680nm) = 0.00000519673$$
$$\beta(550nm) = 0.0000121427$$
$$\beta(440nm) = 0.0000296453$$

On 680, 550 i 440 són la longitud d'ona que s'aproxima a vermell, verd i blau.

Aquests números representen la proporció de llum que es perd per una única interacció amb una partícula. Si suposem que un raig de llum té una intensitat inicial I_0 i travessa un segment de l'atmosfera amb un coeficient de dispersió β, aleshores la quantitat de llum que no es perd per la dispersió és: [22]

$$I_1 = \underbrace{I_0}_{\text{energia inicial}} - \underbrace{I_0\beta}_{\text{energia perduda}} = I_0\,(1-\beta)$$

Fórmula (18)

Tot i que això és vàlid per a una única col·lisió, ens interessa entendre quanta energia es dispersa a una certa distància. Això vol dir que, en cada punt, la llum restant està sotmesa a aquest procés.

Quan la llum viatja a través d'un medi uniforme amb coeficient de dispersió β, com podem calcular quant sobreviu després de viatjar durant una certa distància?

Sempre que un procés multiplicatiu com $(1-\beta)$ es repeteix en un segment continu, el nombre d'Euler fa la seva gran aparició. La quantitat de llum que sobreviu a la dispersió després de viatjar x metres és:

$$I = I_0 exp\{-\beta x\}$$

Fórmula (19)

Un cop més, ens trobem amb una funció exponencial. Això no està de cap manera relacionat amb la funció exponencial utilitzada per descriure la relació de densitat ρ. La raó per la qual ambdós fenòmens es descriuen mitjançant funcions exponencials és que tots dos estan sotmesos a una decadència exponencial. A més d'això, no hi ha altres connexions entre ells. [22]

6.2.14 TRANSMISSIÓ UNIFORME

Prèviament hem introduït el concepte de transmitància T, com la relació de llum que sobreviu al procés de dispersió després de viatjar per l'atmosfera. Ara tenim tots els elements que necessitem per obtenir finalment una equació que ho descriu.

La quantitat de llum dispersa depèn de la distància recorreguda. Com més llarg sigui el viatge, més forta serà l'atenuació. Segons la llei de la decadència exponencial, la quantitat de llum a I_P es pot calcular de la següent manera: [22]

$$I_P = I_S exp\{-\beta\overline{CP}\}$$

Fórmula (20)

On $\overline{CP}$ és la longitud del segment en metres de C a P, i $exp\{x\}$ és la funció exponencial e^x.

6.2.15 TRANSMISSIÓ ATMOSFÈRICA

Hem basat la nostra equació en el supòsit que la possibilitat de ser desviada (el coeficient de dispersió β) és la mateixa per a cada punt al llarg de $\overline{CP}$, però malauradament, no és així.

El coeficient de dispersió depèn fortament de la densitat atmosfèrica. Més molècules d'aire per metre cúbic significa més possibilitats d'impacte. La densitat de l'atmosfera d'un planeta no és uniforme, però canvia en funció de l'altitud. Això també significa que no podem calcular la dispersió fora de $\overline{CP}$ en un sol pas. Hem de calcular la dispersió exterior en cada punt, utilitzant el seu propi coeficient de dispersió. [22]

Per entendre com funciona això, comencem amb una aproximació. El segment $\overline{CP}$ es divideix en dos, $\overline{CQ}$ i $\overline{QP}$.

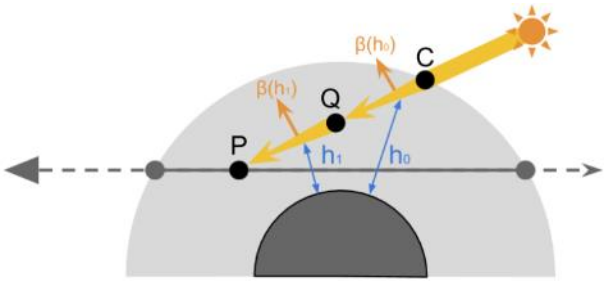

Figura 78 Diagrama de transmissió atmosfèrica. Font: [22]

Calculem primer la quantitat de llum de C que arriba a Q:

$$I_Q = I_S \exp\left\{-\beta(\lambda, h_0)\overline{CQ}\right\}$$

Fórmula (21)

Aleshores, utilitzem el mateix mètode per calcular la quantitat de llum que arriba a P des de Q:

$$I_P = \boxed{I_Q} \exp\left\{-\beta(\lambda, h_1)\overline{QP}\right\}$$

Fórmula (22)

Si substituïm I_Q a la segona equació i simplifiquem, obtenim:

$$I_P = \boxed{I_S \exp\left\{-\beta(\lambda, h_0)\overline{CQ}\right\}} \exp\left\{-\beta(\lambda, h_1)\,\overline{QP}\right\} =$$

$$= I_S \exp\left\{-\beta(\lambda, h_0)\overline{CQ} - \beta(\lambda, h_1)\,\overline{QP}\right\}$$

Fórmula (23)

Si $\overline{CQ}$ i $\overline{QP}$ tenen la mateixa longitud ds, podem simplificar encara més la nostra expressió:

$$I_P = I_S \exp\left\{-\boxed{(\beta(\lambda, h_0) + \beta(\lambda, h_1)\,)ds}\right\}$$

Fórmula (24)

En el cas de dos segments d'igual longitud amb diferents coeficients de dispersió, la dispersió exterior es pot calcular sumant el coeficient de dispersió dels segments individuals, multiplicat per les longituds del segment.

Podem repetir aquest procés amb un nombre arbitrari de segments, apropant-nos cada cop més al valor real. Això condueix a la següent equació:

$$I_P = I_S \exp\left\{-\boxed{\sum_{Q\in\overline{CP}} \beta\left(\lambda, h_Q\right) ds}\right\}$$

Fórmula (25)

On h_Q és l'altitud del punt Q.

El mètode de dividir una línia en diversos segments tal com hem fet s'anomena integració numèrica.
Si suposem que la quantitat inicial de llum rebuda és igual a 1, obtenim l'equació de la transmitància atmosfèrica sobre un segment arbitrari:

$$T\left(\overline{CP}\right) = \exp\left\{-\sum_{Q\in\overline{CP}} \beta\left(\lambda, h_Q\right) ds\right\}$$

Fórmula (26)

Podem ampliar encara més aquesta expressió substituint β pel valor real utilitzat per la dispersió de Rayleigh, β :

$$T\left(\overline{CP}\right) = \exp\left\{-\sum_{Q\in\overline{CP}} \boxed{\frac{8\pi^3\left(n^2-1\right)^2}{3}\frac{\rho\left(h_Q\right)}{N}\frac{1}{\lambda^4}} ds\right\}$$

Fórmula (27)

Molts factors de β són constants i es poden extreure de la suma:

$$T\left(\overline{CP}\right) = \exp\left\{-\underbrace{\frac{8\pi^3\left(n^2-1\right)^2}{3}\frac{1}{N}\frac{1}{\lambda^4}}_{\substack{\text{constant}\\ \beta(\lambda)}} \overbrace{\sum_{Q\in\overline{CP}} \rho\left(h_Q\right) ds}^{\text{optical depth } D\left(\overline{CP}\right)}\right\}$$

Fórmula (28)

La quantitat expressada per la suma es coneix com a profunditat òptica $D(\overline{CP})$, i és el que calcularem realment al codi del shader. La part restant és un coeficient multiplicatiu que correspon al coeficient de dispersió al nivell del mar.

Per resumir-ho:

$$T(\overline{CP}) = exp\{-\beta(\lambda)D(\overline{CP})\}$$

Fórmula (29)

6.2.16 PROFUNDITAT ÒPTICA *(OPTICAL DEPTH)*

Cada punt al llarg del raig visual $\underline{AB}$ aporta la seva pròpia contribució al color final del píxel que estem dibuixant. Aquesta contribució és, matemàticament parlant, la quantitat dins de la suma:

$$I = I_S\,\beta(\lambda)\,\gamma(\theta)\sum_{P\in\overline{AB}}\underbrace{T\left(\overline{CP}\right)T\left(\overline{PA}\right)\rho(h)\,ds}_{\text{contribució de llum de L(P)}}$$

Fórmula (30)

Com hem fet anteriorment, intentem simplificar-ho. Podem ampliar encara més l'expressió anterior, substituint T per la seva definició real:

$$T\left(\overline{XY}\right) = \exp\left\{-\beta(\lambda)\,D\left(\overline{XY}\right)\right\}$$

Fórmula (31)

El producte de la transmitància sobre $\overline{CP}$ i $\overline{PA}$ esdevé:

$$T\left(\overline{CP}\right)T\left(\overline{PA}\right) =$$

$$= \underbrace{\exp\left\{-\beta(\lambda)\,D\left(\overline{CP}\right)\right\}}_{T\left(\overline{CP}\right)}\underbrace{\exp\left\{-\beta(\lambda)\,D\left(\overline{PA}\right)\right\}}_{T\left(\overline{PA}\right)} =$$

$$= \exp\left\{-\beta(\lambda)\left(D\left(\overline{CP}\right) + D\left(\overline{PA}\right)\right)\right\}$$

Fórmula (32)

La transmitància combinada es modela com una disminució exponencial on el coeficient és la suma de les profunditats òptiques al llarg del camí recorregut per la llum $\underline{CP}$ i $\underline{PA}$, multiplicat pel coeficient de dispersió al nivell del mar (β amb h=0).

La primera quantitat que comencem a calcular és la profunditat òptica per al segment PA, que va des del punt d'entrada a través de l'atmosfera fins al punt que estem mostrant actualment. Recordem la definició de profunditat òptica:

$$D\left(\overline{PA}\right) = \sum_{Q\in\overline{PA}}\exp\left\{-\frac{h_Q}{H}\right\}ds$$

Fórmula (33)

6.2.17 SHADER FINAL

Finalment podem definir un *shader* amb codi que pugui descriure tots aquests fenòmens que hem vist.

El primer que farem serà definir la funció amb la que ho calcularem. Cal declarar el primer punt (*inScatterPoint*) on calcularem la dispersió i executem un bucle amb el número de punts que volem (*inScatterPoints*).

Dins del bucle mourem el punt a traves del raig de visió sumant-li la distancia entre punts (*stepSize*). Que podem calcular fàcilment.

```
float calculateLight(vec3 rayOrigin, vec3 rayDir, float rayLength) {
  vec3 inScatterPoint = rayOrigin;
  float stepSize = rayLength / (inScatterPoints - 1.0);

  for (float i = 0.; i < inScatterPoints; i++) {
    inScatterPoint += rayDir * stepSize;
  }
}
```

Per fer un seguiment de la quantitat total de llum dispersa en tots aquests punts, definim la variable *inScatteredLight* inicialitzada a 0. Com podem calcular la quantitat de llum de l'estrella arriba a cada un d'aquests punts? Ja que una part es dispersarà abans de poder arribar i la quantitat que es dispersa depèn de la densitat de l'atmosfera a través d'aquest raig per on la llum viatja, és a dir, la profunditat òptica.

En el codi caldrà conèixer la llargada d'aquest raig que travessa l'atmosfera.

La proporció de llum que arriba fins al punt de dispersió s'anomena transmitància. Observant la següent gràfica, quan la profunditat òptica és zero, la transmitància serà 1 perquè tota la llum arribarà al seu destí. Però a mesura que la profunditat òptica augmenta, més i més llum es dispersarà i així veurem un caiguda exponencial de la quantitat de llum transmesa, la funció real representada aquí és:

$$transmitància = e^{-profunditatÒptica}$$
$$e \approx 2.71828$$

Fórmula (34)

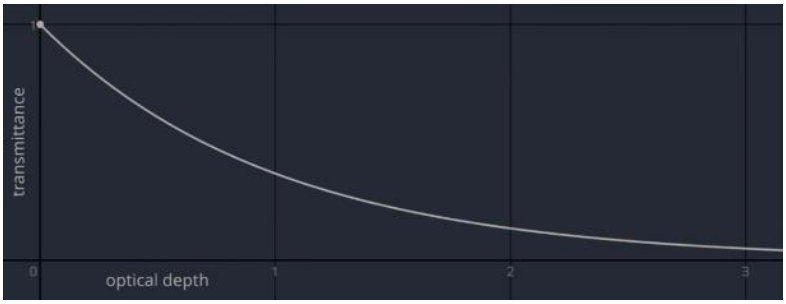

Figura 79 **Transmitància en funció de la profunditat òptica. Font: [25]**

Això es pot representar fàcilment en el codi, on podem escriure-ho utilitzant la funció exponencial integrada que només augmenta *e* a la potència donada.

```
float calculateLight(vec3 rayOrigin, vec3 rayDir, float rayLength) {
  vec3 dirToSun = normalize(sunPosition);
  vec3 inScatterPoint = rayOrigin;
  float stepSize = rayLength / (inScatterPoints - 1.0);
  float inScatteredLight = 0;

  for (float i = 0.; i < inScatterPoints; i++) {
    float sunRayLength = raySphere(planetPosition, atmosphereRadius, inScatterPoint,
dirToSun).y;
    float sunRayOpticalDepth = opticalDepth(inScatterPoint, dirToSun, sunRayLength);
    float transmittance = exp(-sunRayOpticalDepth);
    inScatterPoint += rayDir * stepSize;
  }
}
```

Ara que tenim una idea de quanta llum arribarà al punt, la següent pregunta és quina quantitat es dispersarà cap a la càmera. El factor més important és la densitat al punt de dispersió, de manera que al codi farem ús d'una nova funció per calcular la densitat en un sol punt. Com més gran sigui aquesta densitat, més llum es dispersarà, de manera que podem sumar aquest valor a la llum dispersada, multiplicar-la per la transmitància i la mida del pas.

Una cosa més que hem de tenir en compte és que a mesura que la llum dispersada internament viatja cap a la càmera, una part es dispersarà de nou i pot acabar finalment en la càmera. Així que al codi calculem la profunditat òptica des del punt de dispersió fins a la càmera i després actualitzem la transmitància per tenir en compte que aquest càlcul. Aquest càlcul es pot simplificar per utilitzar només un exponent únic.

```
float calculateLight(vec3 rayOrigin, vec3 rayDir, float rayLength) {
  vec3 dirToSun = normalize(sunPosition);
  vec3 inScatterPoint = rayOrigin;
  float stepSize = rayLength / (inScatterPoints - 1.0);
  float inScatteredLight = 0.;

  for (float i = 0.; i < inScatterPoints; i++) {
    float sunRayLength = raySphere(planetPosition, atmosphereRadius, inScatterPoint,
dirToSun).y;
    float sunRayOpticalDepth = opticalDepth(inScatterPoint, dirToSun, sunRayLength);
    float viewRayOpticalDepth = opticalDepth(inScatterPoint, -rayDir, stepSize * i);
    float transmittance = exp(-(sunRayOpticalDepth + viewRayOpticalDepth));
    float localDensity = densityAtPoint(inScatterPoint);
    inScatteredLight += localDensity * transmittance * stepSize;
    inScatterPoint += rayDir * stepSize;
  }
  return inScatteredLight;
}
```

Encara hem d'implementar les dues funcions que hem cridat però no existeixen, així que comencem amb la que calcula la densitat en un punt.

Recordem que l'atmosfera és més densa a prop de la superfície i es torna exponencialment menys densa com més amunt anem[48], així que al codi i calculem l'alçada sobre la superfície i escalem el resultat perquè sigui 0 a la superfície i 1 a la capa exterior de l'atmosfera per facilitar-ne el treball.

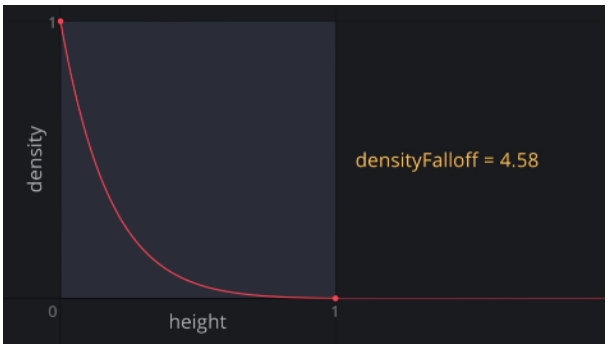

Figura 80 **Densitat** atmosfèrica en funció de **l'altura. Font: [25]**

Ara podem utilitzar la mateixa funció exponencial d'abans i multiplicar l'alçada per una variable per poder controlar la forma d'aquesta corba.

```
float densityAtPoint(vec3 densitySamplePoint) {
  float heightAboveSurface = length(densitySamplePoint - planetPosition) - planetRadius;
  float height01 = heightAboveSurface / (atmosphereRadius - planetRadius);
  float localDensity = exp(-height01 * densityFalloff) * (1.0 - height01);

  return localDensity;
}
```

Passem a la funció de profunditat òptica, que es tracta essencialment de calcular la densitat atmosfèrica mitjana al llarg d'un raig de llum, de manera que estimem el valor simplement dividint-lo en una sèrie de punts calculant la densitat a cada un d'ells amb la funció que acabem d'escriure i després cal sumar-los i multiplicar-los per la mida del pas.

Establim el punt de mostra perquè comenci a l'origen del raig. Després calculem la mida del pas i inicialitzem la variable amb la profunditat òptica a zero. A continuació fem un bucle pel nombre de punts que volem calcular i afegim el resultat a la profunditat òptica multiplicada per la mida del pas. Finalment, movem el punt actual al següent punt del raig i retornem el resultat un cop acabi el bucle.

```
float opticalDepth(vec3 rayOrigin, vec3 rayDir, float rayLength) {
  vec3 densitySamplePoint = rayOrigin;
  float stepSize = rayLength / (opticalDepthPoints -1.0);
  float opticalDepth = 0.0;

  for (float i = 0.; i < opticalDepthPoints; i++) {
    float localDensity = densityAtPoint(densitySamplePoint);
    opticalDepth += localDensity * stepSize;
    densitySamplePoint += rayDir * stepSize;
  }

  return opticalDepth;
}
```

Fet això, podem tornar al programa principal i afegirem una comprovació per saber si estem mirant a través de l'atmosfera. Si és així, calcularem quin és el primer punt del raig que hi ha dins de l'atmosfera i podem cridar a la funció per calcular la llum que passa fins a aquest punt juntament amb la direcció del raig i la distància a través de l'atmosfera. Finalment retornarem el color original barrejat amb la llum, si no estem mirant a través de l'atmosfera, només retornarem el color l'original.

```
void main() {
  vec4 originalCol = texture(tDiffuse, vUv);
  ...
  if (dstThroughAtmosphere > 0.0) {
        vec3 pointInAtmosphere = rayOrigin + rayDirection * dstToAtmosphere;
        float light = calculateLight(pointInAtmosphere, rayDir, dstThroughAtmos);
        gl_FragColor = originalCol * (1.-light) + light;
  }
}
```

Encara que hem vist com funcionen molts dels components que defineixen l'aparença final de l'atmosfera, no hem vist com definir exactament la mida de l'atmosfera respecte al planeta, i és que una atmosfera no acaba bruscament a una alçada determinada, sinó que es torna progressivament menys densa amb l'altitud, com hem pogut veure.

Alguns elements que determinen la mida de l'atmosfera són els gasos que la formen, la temperatura i la gravetat del planeta entre altres.[52] Aquest camí pot ser tediós i no assegurar-nos un resultat acceptable en l'entorn de la simulació. És per això que treballarem amb valors que ens donin resultats realistes, sacrificant precisió.

L'atmosfera terrestre té un gruix equivalent, si comprimim l'atmosfera fins a una pressió i densitat uniformes, tindrem uns 10 quilòmetres. Comparant això amb el radi de la Terra, 6.370 quilòmetres[51], el gruix de l'atmosfera terrestre és aproximadament el 0,17% del seu radi. Si tenim en compte el "límit exterior" de l'atmosfera neutra de la Terra, el que anomenem *exobase*, que arriba a uns 600 quilòmetres d'altitud, el gruix equivalent de l'atmosfera és només el 10% del radi de la Terra.[50]. Depenent de com es defineixin les diferents capes que formen l'espai al voltant de la Terra, la definició d'aquest límit podria variar considerablement.

Per tant ens basarem en aquestes dades per reproduir resultats visualment propers a la realitat.

Arribats a aquest punt podem provar a executar el programa i veurem un resultat com el de la imatge següent, una petita boira al voltant del planeta, el següent pas serà ampliar el codi per implementar la dispersió dels colors:

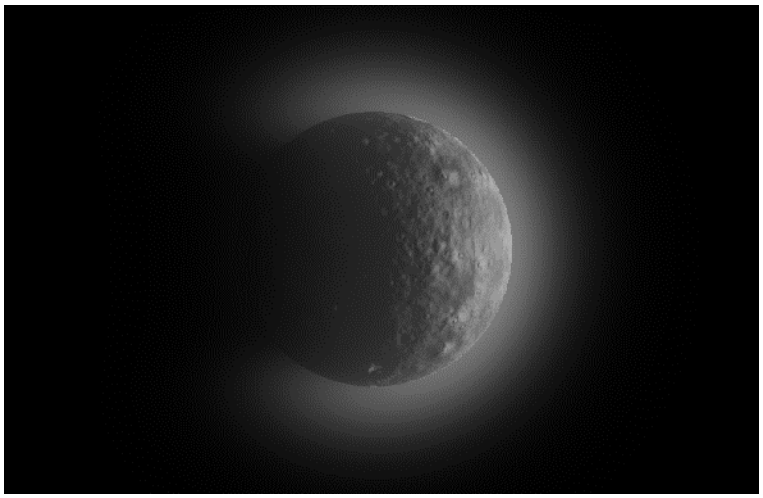

Figura 81 Atmosfera resultant en executar el programa. Font: pròpia

Hem de formar la nostra imatge amb els colors vermell, verd i blau, així que cal fer nos una idea aproximada d'aquestes longituds d'ona. El vermell s'apropa als 700 nanòmetres, el verds a uns 530 i el blau 440.

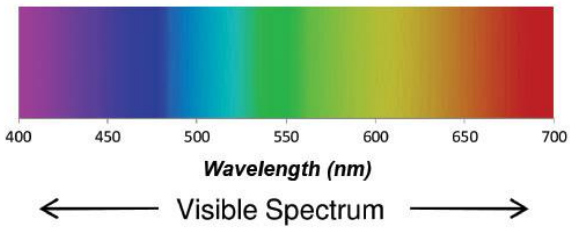

Figura 82 Longitud d'ona λ (nm) de la llum visible de l'espectre electromagnètic. Font: [24]

El primer que farem sera enviar al shader una variable mitjançant uniforms per emmagatzemar aquestes longituds d'ona juntament amb algunes altres variables per poder controlar la força de dispersió, la quantitat de punts que volem calcular al raig i de profunditat òptica, així com altres dades necessàries per realitzar els càlculs.

```
this.material.uniforms.tDiffuse.value = this._target.texture;
this.material.uniforms.tDepth.value = this._target.depthTexture;

this.material.uniforms.inverseProjection.value = this.app.camera.projectionMatrixInverse;
this.material.uniforms.inverseView.value = this.app.camera.matrixWorld;
this.material.uniforms.cameraPosition.value = this.app.camera.position;

this.material.uniforms.atmosphereRadius.value = (1 + this.size) * this.app.planet.size;
this.material.uniforms.densityFalloff.value = this.densityFalloff;
this.material.uniforms.opticalDepthPoints.value = this.opticalDepthPoints;
this.material.uniforms.inScatterPoints.value = this.inScatterPoints;

var scatterR = Math.pow(400 / this.waveLengths.x, 4) * this.scatteringStrength;
var scatterG = Math.pow(400 / this.waveLengths.y, 4) * this.scatteringStrength;
var scatterB = Math.pow(400 / this.waveLengths.z, 4) * this.scatteringStrength;
var scatteringCoefficients = new THREE.Vector3(scatterR, scatterG, scatterB);
this.material.uniforms.scatteringCoefficients.value = scatteringCoefficients;
```

Cal recordar que la quantitat de dispersió és inversament proporcional a la quarta potència de la longitud d'ona (Fórmula (12)), llavors podem dir que la força de dispersió de la llum vermella és igual a 1 entre la longitud d'ona multiplicat per 4 al quadrat multiplicada per la força de dispersió.

Farem el mateix per al verd i el blau. Com que no volem que aquests valors siguin ridículament petits perquè és molest treballar-hi, així que en comptes de dividir 1 entre la longitud d'ona, farem servir 400, cosa que farà que els resultats s'apropin més a 1.

Aquestes dades les enviem al *shader* i ara cal adaptar-lo per fer ús d'aquestes noves dades. A la funció *calculateLight()*, on tenim la nostra variable de transmitància, en comptes de ser de tipus *float*, indicarem *vec3* perquè pugui contenir diferents valors de transmitància per a la llum vermella, verda i blava.

També multipliquem la variable de profunditat òptica dins de la funció exponencial per el coeficient de dispersió. Si mirem un gràfic de la transmitància amb la força de dispersió proper a 0, pràcticament es transmetrà tota la llum.

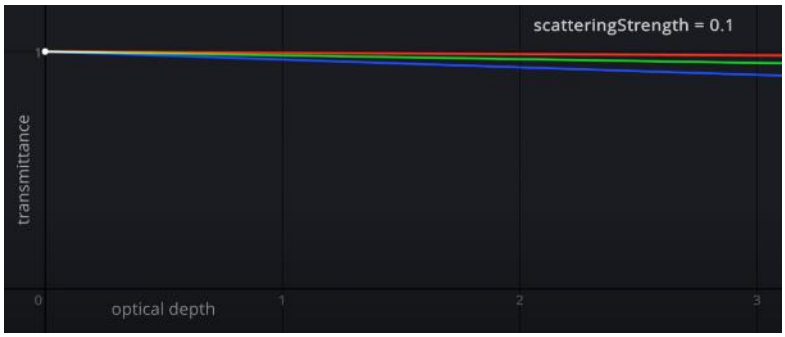

Figura 83 Gràfic de la funció de transmitància on la força de dispersió és 0.1. Font: [25]

A mesura que augmenta la força, es pot veure que les diferents longituds d'ona es separen en funció del coeficient de dispersió que hem calculat i observem que la llum vermella és la que més es transmet i la llum blava la que menys, així que les matemàtiques semblen funcionar.

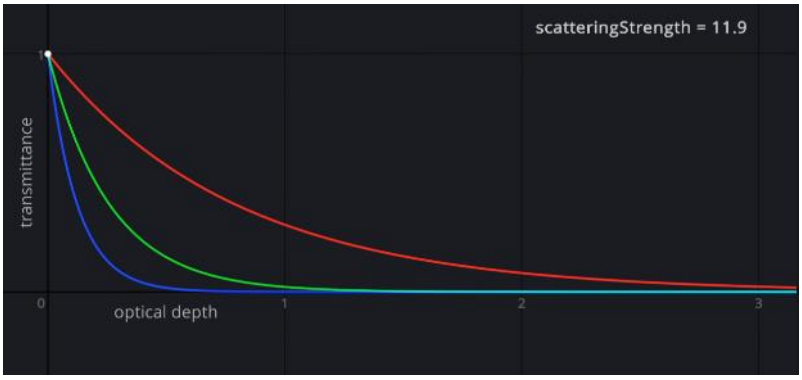

Figura 84 Gràfic de la funció de transmitància on la força de dispersió és 11.9. Font: [25]

El coeficient de dispersió també afecta a la quantitat de llum que es dispersa cap a la càmera, així que haurem de comptabilitzar-la a la variable *inScatteredLight*. Això implica que la variable *inScatteredLight* haurà de ser un vec3 i, per tant, la funció *calculateLight* també haurà de tornar un vec3.

També afegirem un paràmetre a la funció per obtenir el color original i així poder barrejar-lo amb la llum dispersada. Després d'actualitzar i ajustar la resta de funcions perquè funcioni amb aquesta

nova configuració, estem preparats per a provar el programa. La funció *calculateLight* final és la següent:

```
vec3 calculateLight(vec3 rayOrigin, vec3 rayDir, float rayLength, vec3 originalCol) {
  vec3 dirToSun = normalize(sunPosition);
  vec3 inScatterPoint = rayOrigin;
  float stepSize = rayLength / (inScatterPoints - 1.0);
  vec3 inScatteredLight = vec3(0.0);
  float viewRayOpticalDepth = 0.0;

  for (float i = 0.; i < inScatterPoints; i++) {
    float sunRayLength = raySphere(planetPosition, atmosphereRadius, inScatterPoint,
dirToSun).y;
    float sunRayOpticalDepth = opticalDepth(inScatterPoint, dirToSun, sunRayLength);
    viewRayOpticalDepth = opticalDepth(inScatterPoint, -rayDir, stepSize * i);
    vec3 transmittance = exp(-(sunRayOpticalDepth + viewRayOpticalDepth) *
scatteringCoefficients);
    float localDensity = densityAtPoint(inScatterPoint);

    inScatteredLight += localDensity * transmittance * scatteringCoefficients * stepSize;
    inScatterPoint += rayDir * stepSize;
  }
  float originalColTransmittance = exp(-viewRayOpticalDepth);
  return originalCol * originalColTransmittance + inScatteredLight;
}
```

En executar el codi podem veure l'atmosfera sobre el planeta, aquesta vegada amb uns colors molt propers a la realitat gràcies als nostres càlculs. Podem modificar els paràmetres de l'atmosfera per veure l'impacte que té en temps real (Mida, corba de densitat, quantitat de punts als raigs de llum o fins i tot modificar els paràmetres de les longituds d'ona per modificar el color de l'atmosfera).|

Figura 85 Resultat final del model atmosfèric en execució. Font: pròpia

CAPÍTOL 7. CLIMA GLOBAL

En aquest capítol s'introdueixen els principis del funcionament del clima a escala global en un planeta com la Terra, i es mostren les aproximacions necessàries per poder implementar en el nostre programa aspectes com la pressió atmosfèrica, circulació d'aire, humitat, vapor d'aigua i vegetació entre altres característiques que defineixen un sistema climàtic.

7.1 PRINCIPIS BÀSICS DEL CLIMA

Simular el sistema climàtic d'un planeta sencer és notòriament complicat, però per sort resulta que es pot aproximar amb relativa facilitat.

Pràcticament tot el que és important sobre els climes es pot deduir dels següents principis físics:

1. Tot el calor prové del sol.
2. L'aigua s'escalfa i es refreda molt més lentament que la terra.
3. L'aire calent puja, l'aire fred s'enfonsa; això és perquè l'aire s'expandeix a mesura que s'escalfa i, per tant, es torna menys dens.
4. L'aire fred dona lloc a zones d'alta pressió, i l'aire calent dona lloc a zones de baixa pressió.
5. El vent es mou de zones d'alta pressió a zones de baixa pressió.
6. A causa de l'efecte Coriolis, l'efecte de la rotació de la terra sobre el flux d'aire, els vents es desvien cap a la dreta a l'hemisferi nord i cap a l'esquerra al sud.
7. L'aire que puja és propici a la causar precipitacions, l'aire que s'enfonsa no ho és. A mesura que l'aire puja, es refreda i les gotes d'aigua es poden condensar en núvols
8. L'aire calent transporta més humitat que l'aire fred. [55]

Si la Terra no girés i es mantingués estacionària, l'atmosfera circularia entre els pols (zones d'alta pressió) i l'equador (una zona de baixa pressió) en un patró senzill d'anada i tornada. Però com que la Terra gira, l'aire que circula es desvia. En lloc de circular en un patró recte, l'aire es desvia cap a la dreta a l'hemisferi nord i cap a l'esquerra a l'hemisferi sud, donant lloc a camins corbes. Aquesta desviació s'anomena efecte Coriolis i porta el nom del matemàtic francès Gaspard Gustave de Coriolis (1792-1843)[56].

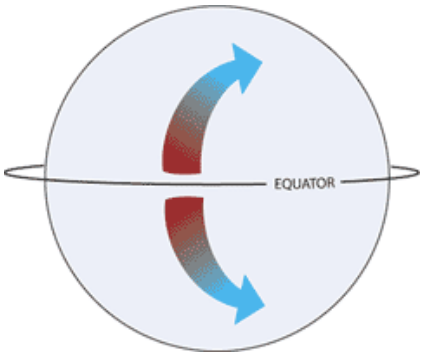

Figura 86 L'efecte Coriolis en acció. Font: [56]

7.2 PRESSIÓ ATMOSFÈRICA

La primera etapa consisteix a localitzar les zones a gran escala d'alta i baixa pressió.

El més important a tenir en compte és el cinturó de baixa pressió anomenat zona de convergència intertropical, o **ITCZ**, sobre el qual les característiques de temperatura i pressió són teòricament simètriques; aquesta zona és causada per l'ascens d'aire tropical calent. [55]

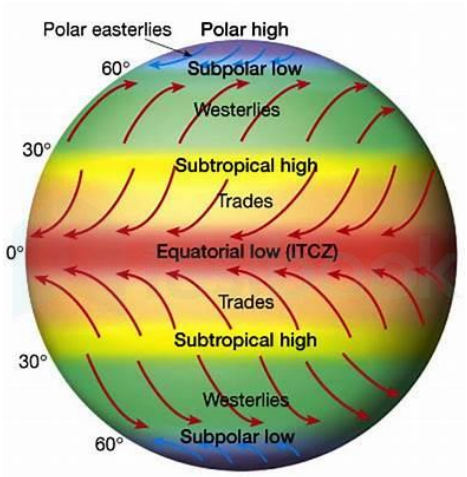

Figura 87 Distribució dels tròpics i la ITCZ al planeta Terra. Font: [57]

Hi ha un altre cinturó d'alta pressió entre el **ITCZ** i els pols, conegut com a zona d'alta pressió subtropical, o **STHZ**, que és causat pel refredament de l'aire del **ITCZ** i que s'enfonsa cap a terra. Entre la **STHZ** i els pols hi ha el front polar o **PF**, una banda de baixa pressió on l'aire fred dels pols es troba amb l'aire càlid de la **STHZ**.

Si la superfície del planeta fos completament aigua uniforme, la distribució d'aquests cinturons de pressió i els vents dominants seria com es pot veure a la Figura 88. [55]

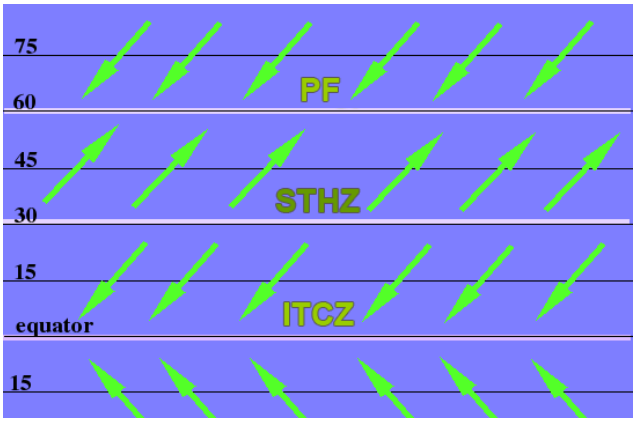

Figura 88 Distribució dels cinturons de pressió. Font: [55]

La presència de terra té dos efectes en la distribució de la pressió, ambdós resultats dels principis 2,3 i 4 que hem vist a l'inici del capítol. Els cinturons de pressió frontal es dobleguen cap al nord sobre la terra al juliol i cap al sud al gener, i es trenquen per les àrees de pressió estacional sobre el terreny. En general, com més gran sigui la superfície de terra, més notable serà l'efecte.

A l'hivern, la baixada de temperatura de la terra crea una zona d'alta pressió a l'interior, que es barreja amb la zona d'alta pressió al voltant de la **STHZ** i deixa sistemes de baixa pressió sobre els oceans:

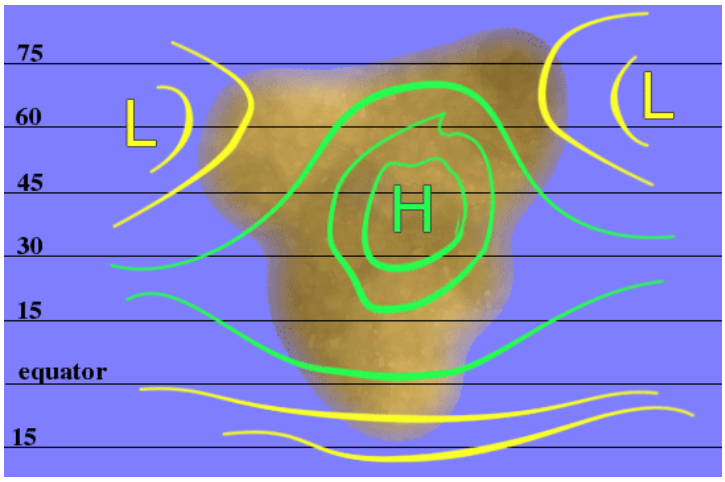

Figura 89 . Zones d'alta i baixa pressió a l'hivern. Font: [55]

Mentre que a l'estiu la terra s'escalfa per crear una zona de baixes pressions, que s'uneix amb la **ITCZ** i la **PF**, deixant zones d'alta pressió sobre els oceans:

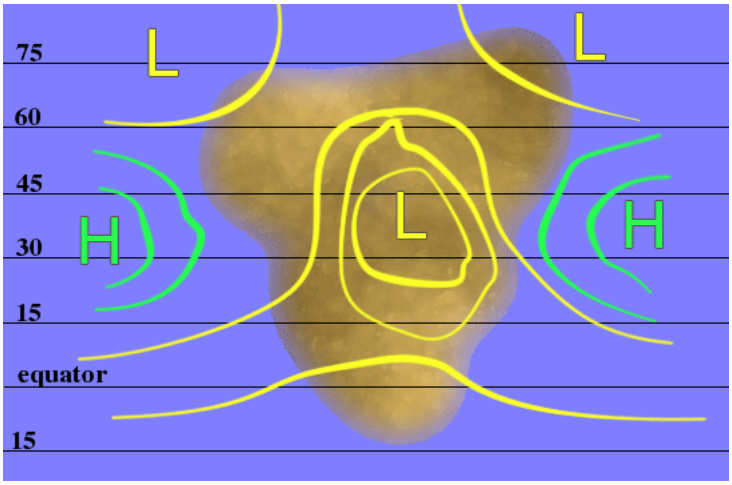

Figura 90 Zones d'alta i baixa pressió a l'estiu. Font: [55]

En general, aquestes zones de pressió es troben a l'est de la meitat longitudinal (est-oest) del continent, i són més intenses quan la massa terrestre circumdant és més gran. Això es nota especialment amb Àsia. En conseqüència, el gradient de pressió és més gran a les costes est que a les costes oest; la diferència precisa depèn de la forma del continent. [55]

Per a crear un mapa de pressió mitjana a nivell del mar (**MSLP** - Mean sea level pressure) cal saber on es troben les formes de terra que hi ha entre els oceans i l'impacte que té la latitud en cada punt. Per definició, la pressió mitjana a nivell del mar del nostre planeta és d'una atmosfera, o 1013.25 hPa.[58]

Si agafem dades d'un mapa MSLP real de la Terra, el separem segons si són zones terrestres o oceàniques i tracem el MSLP en funció de la latitud, acabem amb dues corbes sinusoidals per a la terra i l'oceà amb lleugerament diferents formes. [7]

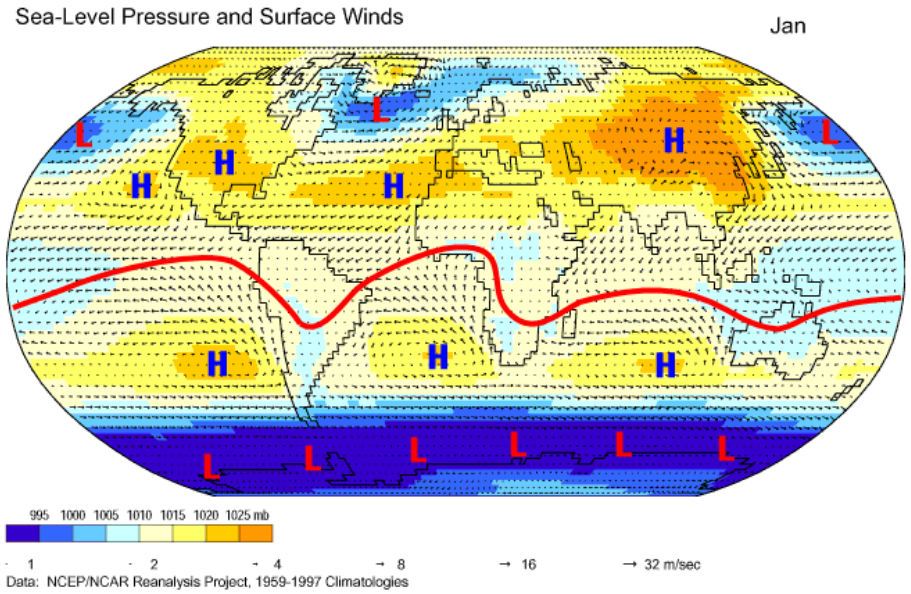

Figura 91 Mapa de vents i pressió atmosfèrica de gener, 1959-1997. La línia vermella representa la ITCZ. Font: [59]

En ajustar els paràmetres adequadament, podem obtenir un model brut de la pressió mitjana anual.

```
if (land) {
    mslp = 1012.5 - 6. * cos(lat*PI/45.);
} else { // ocean
    mslp = 1014.5 - 20. * cos(lat*PI/30.);
}
```

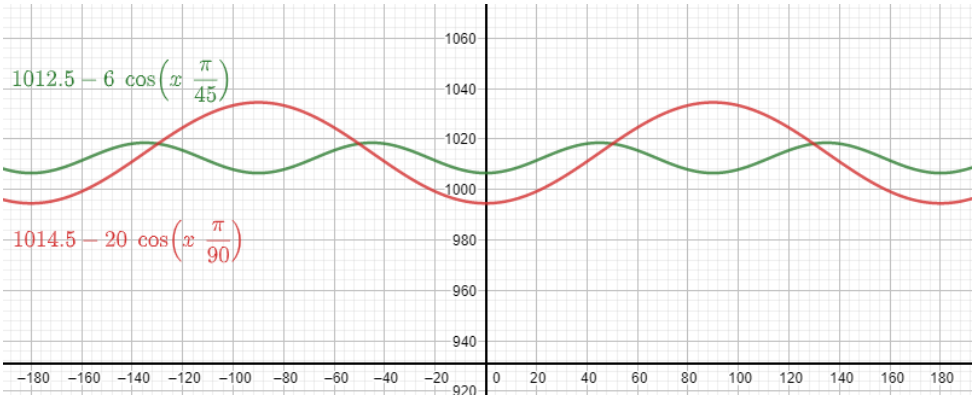

Figura 92 Corbes sinusoidals per a la terra i l'oceà en funció de la latitud. Font: pròpia

Això no és suficient per generar un mapa MSLP realista, ja que generar valors per a la terra i l'oceà per separat provoca discontinuïtats molt brusques als límits entre ells. En realitat, MSLP varia suaument al llarg de la transició de l'oceà a la terra. Aquest procés es pot aproximar bastant bé aplicant simplement un desenfocament al mapa MSLP.

L'eix de rotació de la Terra està inclinat respecte al seu pla orbital uns 23.5 graus. Això és el que provoca les estacions. Per permetre que el clima canviï amb les estacions, també cal modelar la diferència de MSLP entre gener i juliol. Una vegada més, les dades terrestres suggereixen que això segueix un patró sinusoïdal.

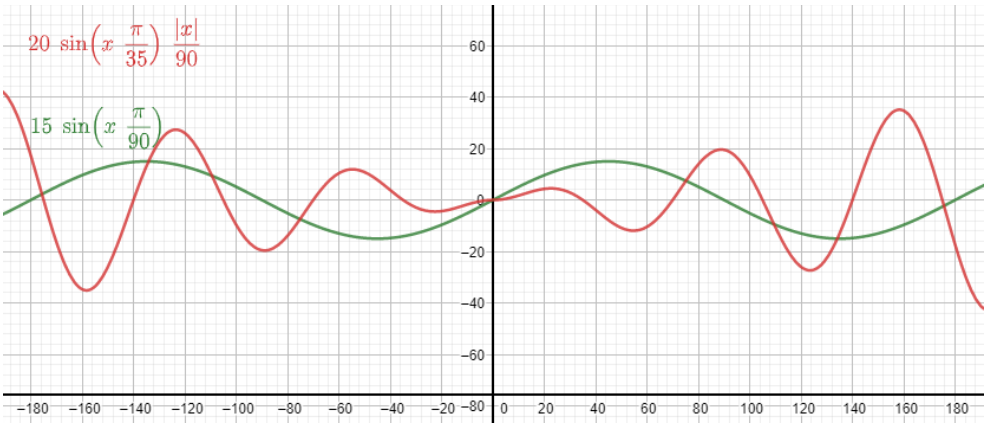

Figura 93 Corbes sinusoidals per representar la diferència de MSLP entre les estacions en funció de la latitud. Font: pròpia

Després d'ajustar els paràmetres i aplicar un desenfocament gaussià, podem combinar-lo amb el mapa MSLP anual per generar patrons climàtics dinàmics que varien al llarg de l'any.

```
if (land) {
	delta = 15. * sin(lat*PI/90.);
} else { // ocean
	delta = 20. * sin(lat*PI/35.) * abs(lat)/90.;
}
```

El desenfocament gaussià que fem servir és un efecte per suavitzar imatges. En essència, l'efecte barreja lleugerament els colors dels píxels que estiguin veïns l'un a l'altre, cosa que provoca que la imatge perdi alguns detalls minúsculs i, així, fa que la imatge es vegi més suau (encara que menys nítida o clara). Utilitza una funció gaussiana (que també expressa la distribució normal en estadístiques) per calcular la transformació que cal aplicar a cada píxel de la imatge. [60]

El codi que fem servir per a definir aquesta distribució normal és el següent:

```
#define SIGMA vec4(6,4,1,0)
vec4 normpdf(float x) {
	return 0.39894 * exp(-0.5 * x*x / (SIGMA*SIGMA)) / SIGMA;
}
```

Després d'aplicar colors càlids a les àrees de baixa pressió (entre 1000 i 1012 hPa) i colors freds a les àrees d'alta pressió (entre 1012 i 1024 hPa), podem veure el resultat:

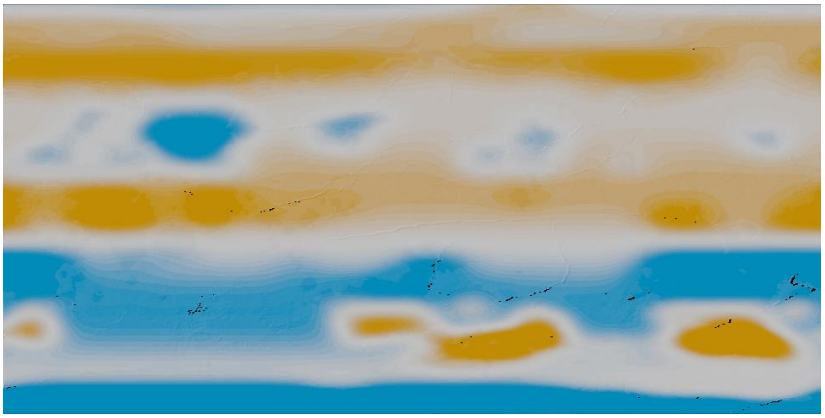

Figura 94 Visualització de la pressió atmosfèrica. Font: pròpia

Per veure l'efecte que te aquest desenfocament, podem desactivar-lo i notar la diferencia:

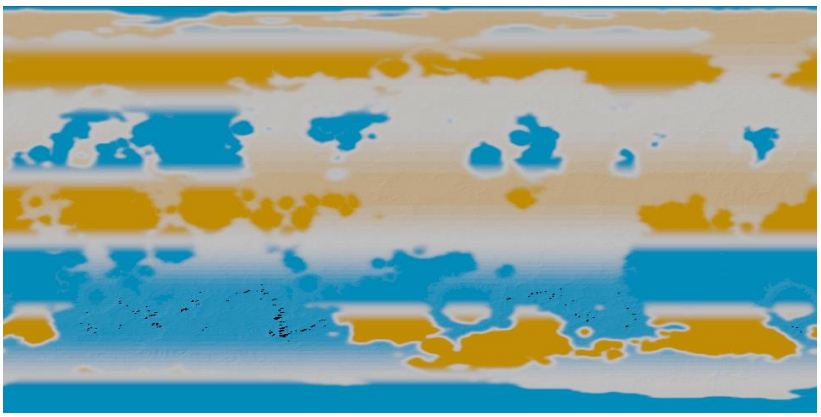

Figura 95 Visualització de la pressió atmosfèrica sense desenfocament. Font: pròpia

Amb això ja tenim una base que, encara que sigui una aproximació extremadament simplificada en comparació a la complexitat que pot arribar a tenir a la realitat, ens servirà per a poder explorar els altres elements que conformen un sistema climàtic.

7.3 CORRENTS DE VENT

Ara que tenim un model de pressió atmosfèrica simple en funcionament, és possible generar corrents de vent i temperatures.

El vent, en termes meteorològics, és un flux d'aire d'una zona d'alta pressió a una zona de baixa pressió. La força (velocitat) del vent augmenta amb la diferència de pressió.

Els vents tenen dos efectes importants sobre el clima: transporten humitat i són la principal causa dels corrents oceànics. Recullen humitat mentre bufen sobre els oceans i la dipositen com a pluja o neu sobre la terra. Òbviament, un vent només pot portar una quantitat finita d'humitat, de manera que s'assecarà després de bufar per una gran àrea de terra.

A causa de l'efecte Coriolis, els vents no bufen directament d'alta pressió a baixa pressió, sinó que es desvien, a l'hemisferi nord, en sentit horari al voltant de zones d'alta pressió i en sentit antihorari al voltant de zones de baixa pressió. A l'hemisferi sud, la desviació és en sentit contrari.[55]

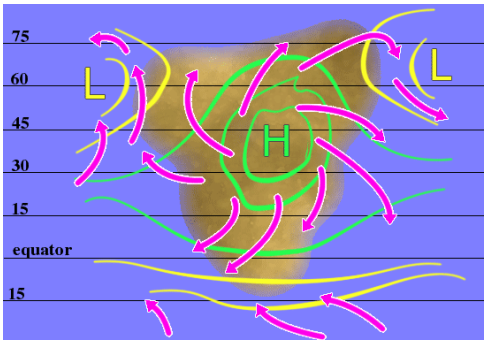

Figura 96 Direcció dels vents dominants a l'hivern. Font: [55]

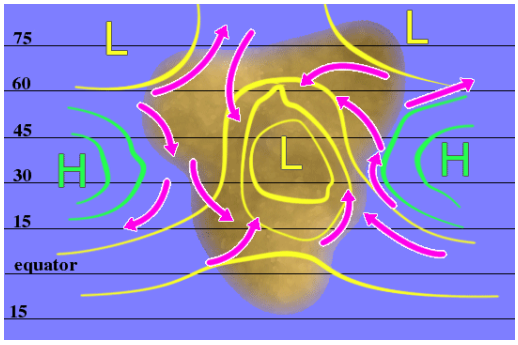

Figura 97 Direcció dels vents dominants a l'estiu. Font: [55]

Per a implementar això, fem us del mapa MSLP per obtenir els vectors de direcció del vent gracies a les diferencies de pressió i temperatura entre diferents punts del mapa, aquest vector es **grad** al codi. La velocitat d'aquest vent anirà directament determinada per la diferencia de pressió.

```
vec2 coriolis = 15. * sin(lat*PI/180.) * vec2(-grad.y, grad.x);
vec2 velocity = coriolis - grad;
```

Tot i que és una aproximació molt superficial, aquesta és capaç de generar patrons de circulació del vent bastant realistes, ja que és possible observar fenòmens naturals que es reprodueixen amb bastant exactitud, almenys en l'escala en la qual treballem.

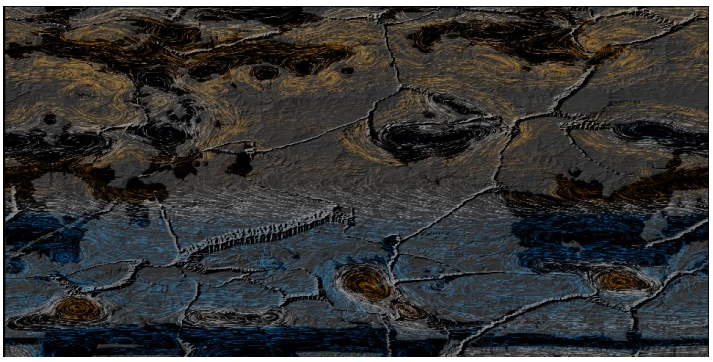

Figura 98 Visualització de les corrents de vent en el programa. Font: pròpia

7.4 TEMPERATURES GLOBALS

La variació anual de la temperatura és l'altre tret característic d'un clima. Com a primera aproximació, la temperatura és més alta a l'equador i disminueix constantment cap als pols.

Les variacions de temperatura són més baixes al llarg de les costes i més altes a les zones allunyades de la influència marítima. La variació augmenta amb la distància dels oceans, i menys amb la distància de la costa oest; les regions orientals dels interiors continentals experimenten així les majors variacions de temperatura.[55]

Això requereix una mica més d'esforç per generar valors propers a la realitat, però amb tot el que hem vist, podem aproximar la temperatura en un punt concret basant-nos en la següent formula:

```
float temp = 40. * tanh(2.2 * exp(-0.5 * pow((lat + 5. * season)/30., 2.))) - 15. -
(mslp - 1012.) / 1.8 + 1.5 * land - 4. * elevation;
```

El resultat és el que podem veure a la Figura 99, amb color blau representant a les àrees amb baixa temperatura (uns -20 °C) i a l'altre extrem en colors més càlids representant temperatures més elevades fins a uns 25 °C. La resta de colors són valors intermedis.

Figura 99 Mapa de la temperatura global en la simulació. Els valors més freds en blau, i els més calents en carabassa. Font: pròpia

En comparació, a la Figura 100 tenim com a exemple un mapa de temperatura de l'aire a la superfície segons dades reals de febrer de 2021.

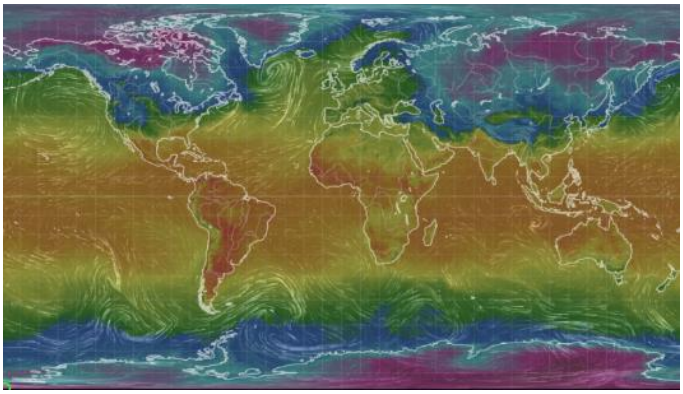

Figura 100 Mapa de la temperatura de la Terra (febrer 2021) amb un rang de temperatures de -47 °C (lila) fins a 38 °C (vermell). Font: [61]

7.5 PRECIPITACIONS I VAPOR D'AIGUA

La precipitació es pot simular a partir del vapor d'aigua de l'oceà, que es transporta a través del vent cap a la terra. La distribució anual de precipitacions en forma de pluja i neu és un dels factors que caracteritzen un clima determinat.

La pluja i la neu apareixen gracies a aquests quatre processos[55]:

- Els vents humits que bufen dels oceans cap a terra.
- Núvols orogràfics, originats dels vents que porten humitat mentre bufen sobre muntanyes i es veuen obligats a pujar. Aquest aire es refreda a mesura que puja, dipositant la seva humitat a les muntanyes.
- Convecció deguda a l'escalfament de l'aire, ja que aquest es refreda a mesura que puja i perd la seva humitat.
- Elevació de l'aire al llarg del front polar (*frontal lifting*). Aquí l'aire càlid de la STHZ és aixecat per l'aire més fred dels pols, provocant les zones de baixa pressió.

Els corrents freds refreden i estabilitzen l'aire, inhibint la formació de precipitacions, mentre que els corrents càlids l'escalfen i el desestabilitzen, afavorint la precipitació[55].

Per a simular l'increment de vapor d'aigua en un punt determinat mitjançant l'evaporació, primer tenim en compte que sigui un punt a l'oceà, llavors apliquem soroll FBM (*Fractional Brownian Motion*) perquè el resultat no sigui massa homogeni. La quantitat de vapor d'aigua dependrà de la temperatura, pressió atmosfèrica i velocitat del vent en aquest punt.

```
if (ocean) wVapour += noise * clamp(temp + 2., 0., 100.)/32. * (0.075 - 3. * div -
0.0045 * (mbar - 1012.));
```

En el cas de la precipitació i els núvols orogràfics, podem simplement reduir aquesta quantitat de vapor d'aigua progressivament en funció del temps per al primer cas, i en funció de l'altura del terreny per al segon cas.

```
w -= 0.005 * w;
w -= 0.3 * length(hgrad);
```

7.6 BIOMA

Com a últim detall, podem considerar l'impacte que te la temperatura i humitat a l'hora de formar els paisatges terrestres i la vegetació. A la Figura 101 podem veure els tipus de creixement que tenen diferents cultius en unes condicions de temperatura concretes.

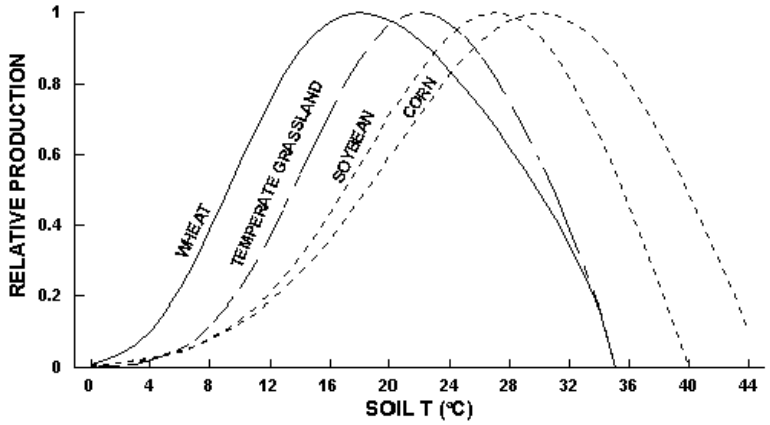

Figura 101 Impacte de la temperatura del sòl en el creixement de les plantes de diferents cultius. Font: [62]

Basant-nos en aquestes dades, podem definir una funció per aproximar la velocitat de creixement de la vegetació en un punt determinat segons la temperatura i humitat.

```
float plant_growth(float moisture, float temp) {
    float growth = clamp(moisture / 3., 0., 1.);
    growth *= smoothstep(0., 25., temp) - smoothstep(25., 35., temp);
    return growth;
}
```

Per a representar amb certa precisió el paisatge del terreny també cal tenir en compte principalment aspectes com l'elevació, quantitat de precipitacions anuals i latitud. Això té un gran impacte en el tipus de bioma que tindrà una àrea determinada, com es pot veure a la Figura 102.

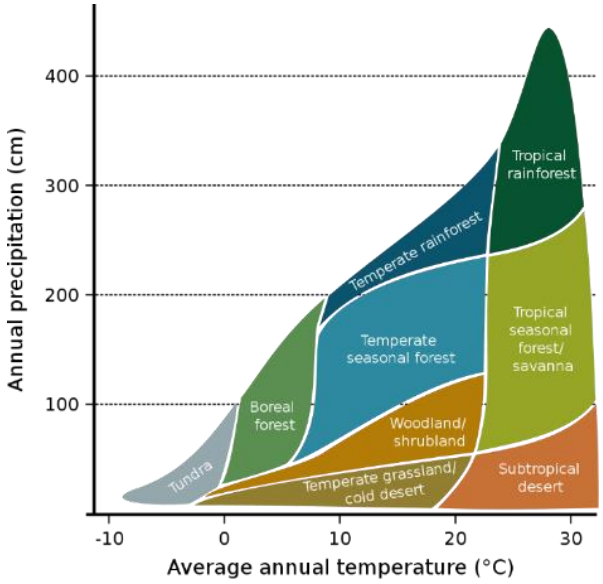

Figura 102 Influència de factors geogràfics climàtics en els ecosistemes. Font: [63]

Podem aproximar aquestes dades d'una forma molt senzilla en funció de la temperatura i la funció de creixement de vegetació que hem definit prèviament.

```
// Dry Land
vec3 dry = vec3(0.89, 0.9, 0.89);
dry = mix(dry, vec3(0.11, 0.10, 0.05), smoothstep(-10., 0., temp));
dry = mix(dry, vec3(1.00, 0.96, 0.71), smoothstep( 0., 20., temp));
dry = mix(dry, vec3(0.81, 0.48, 0.31), smoothstep(20., 30., temp));

// Vegetation
vec3 veg = vec3(0.89, 0.9, 0.89);
veg = mix(veg, vec3(0.56, 0.49, 0.28), smoothstep(-10., 0., temp));
veg = mix(veg, vec3(0.18, 0.34, 0.04), smoothstep( 0., 20., temp));
veg = mix(veg, vec3(0.05, 0.23, 0.04), smoothstep(20., 30., temp));

float moisture = texture(iChannel3, uv).w;
vec4 land = vec4(0,0,0,1);
land.rgb = mix(dry, veg, plant_growth(moisture, temp));
```

D'aquesta forma podem obtenir uns resultats bastant propers a la realitat, almenys al nivell de precisió al que treballem.

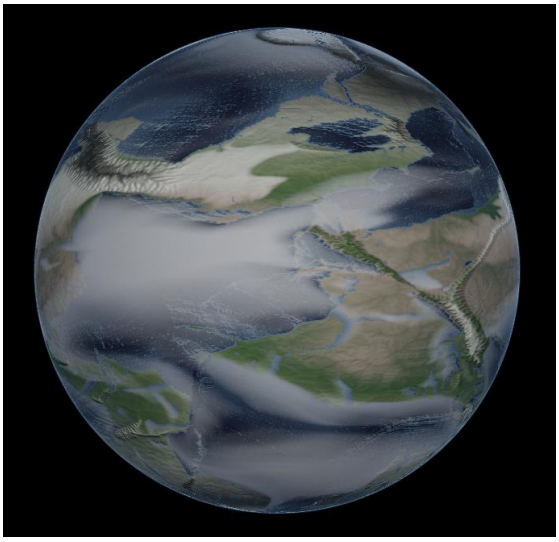

Figura 103 Resultat final en executar el programa. Font: pròpia

Figura 104 Imatge real de la Terra. Font: [53]

7.7 ALTRES CONSIDERACIONS

Els principis descrits fins ara haurien de funcionar prou bé per a un planeta semblant a la Terra. Però, que passa amb altres planetes que puguin ser semblants, però amb certes diferències?

- Si el planeta gira en sentit contrari
  No tindria pràcticament ningun impacte, nomes caldrà intercanvieu "est" i "oest".

- Si el planeta gira molt ràpid
  Les tres bandes de vents dominants a cada hemisferi es deuen a la velocitat de rotació del planeta. Per sobre d'una certa velocitat de rotació, els tres passaran a ser cinc (no poden arribar a ser quatre), i entre l'STHZ i el PF apareixerà una altra corretja cadascuna de baixa pressió i alta pressió.

  Una rotació més ràpida també comportarà dies i nits més curts, que sens dubte tindran altres conseqüències.

- Si el planeta té una inclinació axial inferior
  El moviment nord-sud dels cinturons de pressió serà corresponentment menor, i les zones més petites estaran subjectes als climes que experimenten canvis estacionals; els intervals de temperatura anuals també seran menors. S'afavorirà la selva tropical, els climes marítims, desèrtics càlids i la capa de gel.

- Si el planeta té una inclinació axial superior
  Passarà al contrari que a l'anterior; els efectes estacionals s'incrementaran i les zones subjectes a climes de selva tropical, marítima, desèrtica càlida i casquets glacials seran menors. Una inclinació prou gran (uns 40 graus) eliminarà per complet aquests climes. [55].

CAPÍTOL 8. MISCEL·LÀNIA

En aquest capítol s'inclouen alguns elements que conformen el programa i val la pena mencionar, però no tenen tant impacte o quantitat de contingut com per a dedicar capítols sencers a la seva explicació.

8.1 INTERFÍCIE D'USUARI

La interfície és una peça clau perquè l'usuari pugui interactuar amb el programa i per a poder mostrar informació rellevant. En executar el programa, el primer que apareix és una finestra on es pot seleccionar una escena, la resolució a la qual s'executarà el programa i el *seed*, que és simplement un valor per a gestionar la generació aleatòria de valors. Això significa que si s'executa el programa amb un mateix *seed*, el resultat sempre serà el mateix.

L'escena principal és la primera de la llista, tracta de l'escenari que s'ha presentat en el treball, on es pot visualitzar l'evolució temporal d'un planeta semblant a la Terra. El segon i tercer escenari són exemples més simples del que es pot fer partint de la mateixa base i modificant alguns paràmetres.

Figura 105 Finestra de configuració inicial. Font: pròpia

La creació d'aquest element es realitza principalment a través del *framework* **jQuery UI**, que ens permet crear interfícies de forma ràpida amb una sèrie d'estils i components predefinits. És un *framework* molt senzill pel que fa a la quantitat d'opcions que ofereix, però suficient per a aquesta finalitat. Part del codi *front-end* encarregat de dibuixar la selecció de les escenes és el següent:

```
<legend style="margin-bottom: 0;">Selecciona una escena</legend>
<table id="scene-table" style="width:100%">
  <tr>
    <td>
      <label for="scene-1">Earth-like</label>
      <input value="0" type="radio" name="scene-1" id="scene-1" checked>
    </td>
    <td>
      <p style="font-size: 0.8rem; margin: 0;">Genera un planeta com la terra i
visualitza la seva evolució</p>
    </td>
  </tr>
</table>
```

Part del codi *front-end* encarregat de dibuixar el segon apartat, on es selecciona la resolució de la simulació és el següent:

```
<fieldset>
    <legend style="margin-bottom: 0;">Resolució</legend>
    <p style="font-size: 0.8rem; margin: 0;"><span class="ui-icon ui-icon-alert"
style="float:left; margin:2px 2px 0px 0;"></span>Valors més grans tenen un major
impacte al rendiment.</p>
    <label for="radio-1">256</label>
    <input value="256" type="radio" name="radio-1" id="radio-1">
    <label for="radio-2">512</label>
    <input value="512" type="radio" name="radio-1" id="radio-2">
    <label for="radio-3">1024</label>
    <input value="1024" type="radio" name="radio-1" id="radio-3" checked>
    <label for="radio-4">2048</label>
    <input value="2048" type="radio" name="radio-1" id="radio-4">
    <label for="radio-5">4096</label>
    <input value="4096" type="radio" name="radio-1" id="radio-5">
</fieldset>
```

Per acabar, l'últim apartat on es genera el valor del *seed*. El codi és el següent:

```
<fieldset>
    <legend style="margin-bottom: 0;">Seed</legend>
    <p style="font-size: 0.8rem; margin: 0;">Gestiona la generació aleatòria.</p>
    <input type="text" name="seed-1" id="seed-1" maxlength="10">

    <!-- Random button -->
    <button id="random-seed-button" class="button-small" style="color: #3383BB;
border: none;"><i class='fa-solid fa-dice'></i></button>
</fieldset>
```

Pel que fa al *back-end*, simplement cal inicialitzar el component configurant alguns dels paràmetres disponibles. També es defineixen les accions a realitzar quan s'interactua amb el panell fent clic al botó d'acceptar per a guardar i configurar el programa segons les opcions que es seleccionin en el mateix.

```
$( "#dialog-start" ).dialog({
  dialogClass: "no-close",
  resizable: false,
  height: "auto",
  width: 600,
  modal: false,
  buttons: {
    Ok: function() {
      // Save the values before closing the dialog
      // escena = $(':radio:checked', this)[0].value; //ToDo
      that.app.planet.resolution = $(':radio:checked', this)[1].value;
      that.app.planet.renderScene();
      that.goToPlanet();
      that.loadHistoryInterface();
      $( this ).dialog( "close" );
    }
  }
});
```

En iniciar l'escenari, apareixen els elements de la interfície. A la cantonada superior dreta es troben un parell d'elements. El de la part mes externa es tracta d'un monitor de rendiment per a poder veure fàcilment el rendiment de l'aplicació i detectar problemes durant l'execució d'aquesta.

Figura 106 Interfície del programa amb el panell superior dret ressaltat. Font: pròpia

Novament, es fa ús d'un element extern de la mateixa llibreria de *Three.JS*. El codi principal per a inicialitzar el component i afegir-lo al document és el següent:

```
this.stats = Stats();
this.stats.showPanel( 0 ); // 0: fps, 1: ms, 2: mb, 3+: custom
document.getElementById('stats-container').appendChild(this.stats.domElement);
```

L'altre component és un panell creat amb la llibreria **lil-gui**, que està especialment pensada per permetre afegir i controlar paràmetres de l'escena amb facilitat.

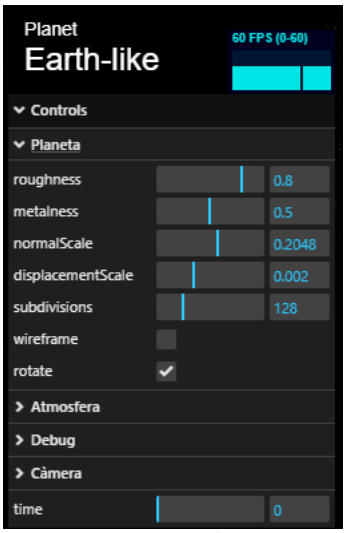

Figura 107 Menú d'opcions desplegat. Font: pròpia

Un cop importada la llibreria, cal inicialitzar l'objecte.En l fer-ho, es poden indicar alguns paràmetres en la crida del constructor.

```
window.gui = new GUI({ autoPlace: false });
```

Es pot omplir aquest panell d'una varietat de components. A continuació, veiem un fragment de codi per crear el menú "planeta". El primer pas és crear una carpeta on es guarden les opcions, com es veu a la primera línia del codi. Seguidament, es creen i defineixen la resta d'opcions, fent referència a la variable que es modifica (aquesta ha d'existir prèviament) i altres paràmetres que permet la llibreria.

```
let matFolder = window.gui.addFolder('Planeta');

matFolder.add(this.app.planet, "roughness", 0.0, 1.0).onChange(value => {
this.app.planet.updateMaterial(); });
matFolder.add(this.app.planet, "metalness", 0.0, 1.0).onChange(value => {
this.app.planet.updateMaterial(); });
matFolder.add(this.app.planet, "normalScale", -1.5, 1.5).listen().onChange(value =>
{ this.app.planet.updateMaterial(); });
matFolder.add(this.app.planet, "displacementScale", -0.05,
0.1).listen().onChange(value => { this.app.planet.updateMaterial(); });
matFolder.add(this.app.planet, "subdivisions", 2, 512, 1).onChange(value => {
this.app.planet.updateGeometry(); });

matFolder.add(this.app.planet, "wireframe").onChange(value => {
this.app.planet.updateMaterial(); });
matFolder.add(this.app.planet, "rotate");

matFolder.close();
```

Com es pot veure, es tracta d'una eina molt fàcil de fer servir, que permet definir ràpidament una interfície bàsica per interactuar amb una aplicació de forma interactiva.

El següent element de la interfície a veure és el panell esquerre, que ofereix una sèrie de botons, cada un amb una funció especifica. El grup de 4 botons que es troben a la part superior gestionen la visibilitat del planeta. El primer, ressaltat en blau, mostra una visió normal del planeta, amb el terreny, núvols i atmosfera. El segon boto activa la visió de plaques tectòniques, amb el mateix resultat que s'ha vist al capítol corresponent. El tercer boto activa la visió de rius i conques hidrogràfiques. El quart boto activa la visió de clima i vent, i per acabar, l'últim botó permet visualitzar la temperatura a cada punt del planeta amb un codi de color.

Figura 108 Interfície del programa amb el panell esquerre ressaltat. Font: pròpia

En aquest cas, s'ha dissenyat manualment a partir de codi HTML estilitzat amb CSS. Per a les icones, es fa servir **fontawesome**, un servei que proporciona un gran catàleg d'icones.

```
<div class="view-panel">
    <button id="normal-view-button" class='btn-interface btn-group active'
title='Vista normal'>
        <i class='fa-solid fa-earth-europe'></i>
    </button>
    <button id="plates-view-button" class='btn-interface btn-group' title='Plaques
tectòniques'>
        <i class='fa-solid fa-globe'></i>
    </button>
    <button id="rivers-view-button" class='btn-interface btn-group' title='Rius'>
        <i class='fa-solid fa-water'></i>
    </button>
    <button id="wind-view-button" class='btn-interface btn-group' title='Clima'>
        <i class='fa-solid fa-wind'></i>
    </button>
    <button id="temperature-view-button" class='btn-interface btn-group'
title='Temperatura'>
        <i class='fa-solid fa-temperature-half'></i>
    </button>
</div>
```

Tots aquests botons tenen un **eventListener** per a definir les accions a realitzar quan es fa clic sobre ells.

```
this.normalViewButton = document.getElementById("normal-view-button");
this.normalViewButton.addEventListener("click", () => {
  this.changeView(0);
  this.normalViewButton.classList.toggle("active");
});
```

El fragment de codi del fitxer d'estils CSS per a donar aquesta aparença als botons és el següent:

```
/* Buttons style  */
.btn-interface {
  position: absolute;
  background-color: rgba(41, 42, 45, 0.6);
  border: none;
  color: rgb(151, 151, 151);
  width: 50px;
  height: 50px;
  font-size: 17px;  /* Icon size */
  cursor: pointer;
  border-radius: 10px;
  justify-content: center;
  align-items: center;
}
.btn-interface:hover, .btn-interface.active { /* Darker background on mouse-over */
  background-color: rgba(65, 105, 225, 0.603);
  color: rgb(255, 255, 255);
}
.btn-group {
  position: relative;
  width: 44px;
  height: 44px;
}
```

Continuant amb la resta dels botons, hi ha una parella a sota del grup anterior. La seva funció es la de modificar el terreny del planeta. El primer boto serveix per a apujar el terreny progressivament i l'altre fa la funció contraria, enfonsar-lo. El funcionament es el següent:
Un cop es fa clic al boto, aquest s'activa i permet fer clic al planeta. Quan es fa clic, es calcula la posició exacta del planeta on s'ha fet clic dins de l'espai 3D de l'escena. Això s'anomena ***raycasting*** i com el indica el nom, consisteix en traçar un raig des de la càmera cap a la posició del ratolí de forma que es detecten els objectes que es troben en la direcció d'aquest.

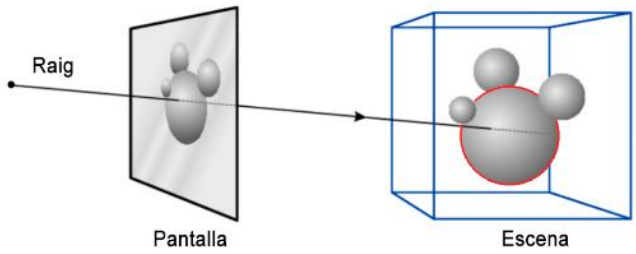

Figura 109 **Concepte de *raycasting*. Font: pròpia**

Aquesta coordenada s'envia al *shader* i es transforma a l'espai 2D d'aquest per a saber on s'ha de modificar el terreny. Finalment, s'aplica la modificació al terreny per a elevar o baixar la seva altura.

```
vec2 r = (p - (uMouse.xy*uResolution.xy)) / brushSize;
float magnitude = 0.;
if (uAddingTerrain) magnitude = 0.5 * exp(-0.5 * dot(r,r));
else if (uRemovingTerrain) magnitude = -1. * (0.5 * exp(-0.5 * dot(r,r)));
if (uMouseClick) gl_FragColor.z += magnitude;
```

A la part inferior es troba l'últim botó. La seva funció és la d'iniciar o posar en pausa la simulació. La seva estructura és similar a la mostrada en els altres botons, però amb una funcionalitat diferent.
Finalment, la resta d'elements de la interfície a veure son els panells superior esquerre i inferior de la pantalla. La seva finalitat es la de informar de l'estat de la simulació a traves d'una línia de temps a la part inferior i un panell de text a la part superior.

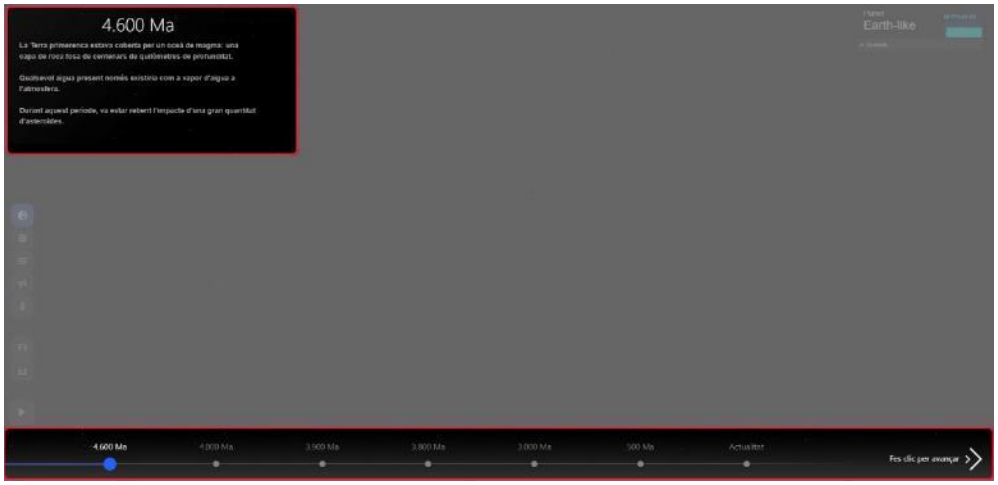

Figura 110 **Interfície del programa amb els panells superior i inferior ressaltat. Font: pròpia**

Aquesta línia de temps està dividida en 7 etapes diferents, indicant en cada una la quantitat de milions d'anys enrere en el temps que representa. Cada cop que es fa clic al botó de la dreta, s'activa la simulació per mostrar l'evolució en cada una de les etapes.

El panell superior també s'actualitza amb la informació rellevant de cada una d'aquestes etapes.

La línia de temps té una estructura i disseny que requereix d'una gran quantitat de codi perquè funcioni. Per sort, existeixen components com ***swiper***, que faciliten aquesta feina incorporant molts d'aquests elements. Un cop importat, cal inicialitzar el component amb alguns paràmetres.

```
var swiper = new Swiper(".swiper", {
  autoHeight: true,
  speed: 8000,
  direction: "horizontal",
  navigation: {
    nextEl: ".swiper-button-next",
    prevEl: ".swiper-button-prev"
  },
  pagination: {
    el: ".swiper-pagination",
    type: "progressbar"
  },
  loop: false,
  effect: "slide",
  spaceBetween: 30,
  on: {
    init: function () {
      $(".swiper-pagination-custom .swiper-pagination-switch").removeClass("active");
      $(".swiper-pagination-custom .swiper-pagination-switch").eq(0).addClass("active");
    },
    slideChangeTransitionEnd: function () {
      $(".swiper-pagination-custom .swiper-pagination-switch").removeClass("active");
      $(".swiper-pagination-custom .swiper-pagination-
switch").eq(swiper.realIndex).addClass("active");
    }
  }
});
```

Un cop configurat, cal definir les accions a cada etapa i de cada botó, així com el disseny del component al fitxer d'estils CSS.

El codi complet està disponible a l'annex 1.

8.2 FONS D'ESTRELLES

Encara que ens volem centrar en el planeta, podem donar una imatge una mica mes realista de l'entorn on es troba afegint un fons d'estrelles.

El que farem és crear una esfera lo suficientment gran per a omplir tot l'escenari. Aquesta tindrà una textura amb aquestes estrelles. Una opció podria ser carregar una imatge en l'esfera, però per seguir amb la temàtica del projecte (*procedural*), generarem amb l'ajut d'una mica de matemàtiques, una textura que ens doni el que busquem.

El soroll Perlin és un tipus de soroll, un algorisme desenvolupat per crear imatges generades per ordinador de manera que semblin més naturals. Va ser desenvolupat per Ken Perlin per tal d'utilitzar-lo a la pel·lícula Tron (1982) [66].

Es pot representar en forma de múltiples dimensions, com veiem a continuació, però en el nostre cas farem servir el soroll en tres dimensions.

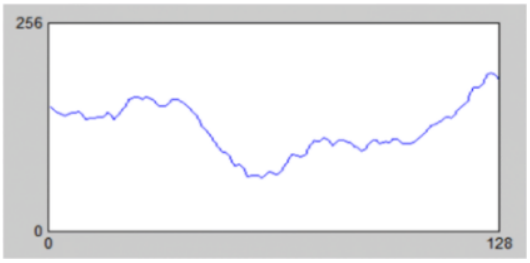

Figura 111 Exemple de soroll perlin 1D. Font: [64]

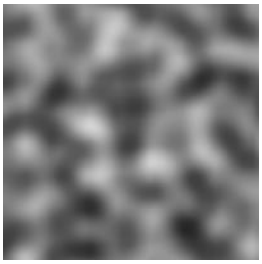

Figura 112 Exemple de soroll perlin 2D. Font: [65]

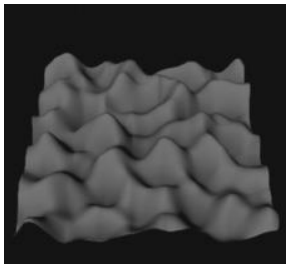

Figura 113 Exemple de soroll perlin 3D. Font: [65]

Podem ajustar alguns paràmetres per a que la funció de soroll ens retorni valors específics.

```
float scale = 2048.;
float density = 20.;
vec3 color = vec3(pow(SmoothNoise(dir*scale), density));
return pow(color*2.25, vec3(power));
```

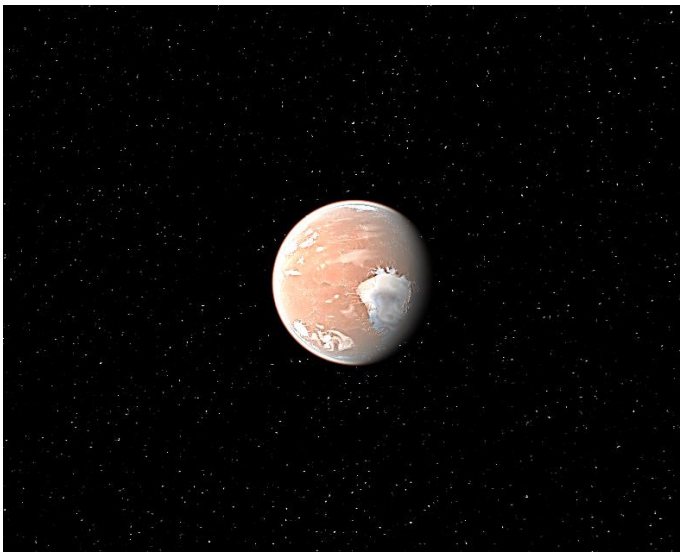

Figura 114 Fons amb les estrelles generades. Font: pròpia

8.3 ESTRELLA DEL SISTEMA PLANETARI

La font de llum que il·lumina el nostre planeta prové d'una estrella, així que podem crear una a partir d'una esfera i aplicar-li un altre *shader* per a donar-li un aspecte similar

Encara que hi han molts tipus d'estrelles amb diferents colors segons la seva temperatura, com podem veure a la Figura 115, crearem una semblant al Sol.

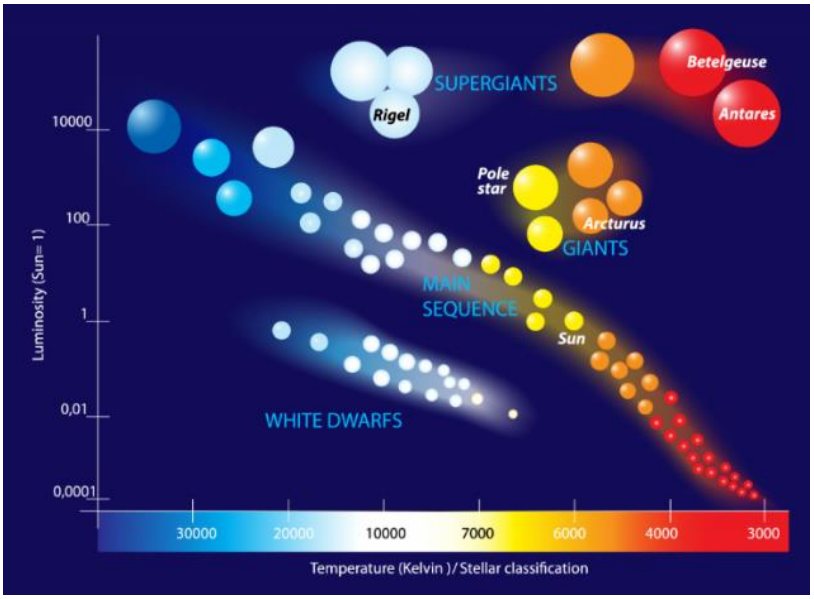

Figura 115 Tipus d'estrelles i el seu color en funció de la temperatura. Font: [67]

No volem només dibuixar una esfera de color groc, farem un degradat per tractar de simular una mica millor com es veu una estrella. El primer que cal fer és enviar la posició en l'escenari, ja que el que farem és dibuixar la textura sempre mirant cap a càmera per a poder crear aquest efecte en una textura en dues dimensions.

També enviarem la distància entre el centre de l'estrella i la càmera.

```
distToCamera = gl_Position.w;
vCenter = projectionMatrix * viewMatrix * vec4( uCenter, 1.0 );
```

Ara calculem la diferència entre vCentre.w i distToCamera. És a dir, la diferència entre la distància a càmera del centre de l'esfera amb la distància a càmera del vèrtex actual.

Així la diferència màxima serà el radi de l'esfera sempre amb independència de la posició de càmera. També cal normalitzar pel radi per poder funcionar amb esferes més grans o petites.

El que queda és cridar a la funció que genera el degradat de color.

```
float radial = distToCamera - vCenter.w;
radial *= (2./uSize);

float brightness = 1. + (radial * 1.2);

gl_FragColor.rgb = brightnessToColor(brightness)*radial;
gl_FragColor.a = radial;


vec3 brightnessToColor(float b) {
    b *= 0.25;
    return (vec3(b, b*b, b*b*b*b)/0.25)*0.5;
}
```

Després d'ajustar els paràmetres fins a trobar uns que s'ajustin al que volem, aquest és el resultat:

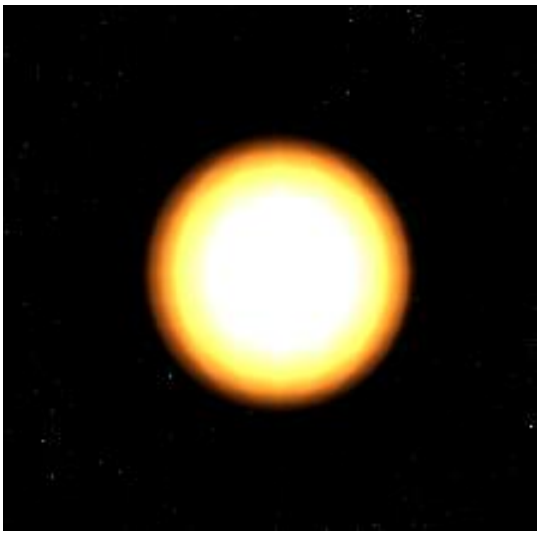

Figura 116 Estrella a l'escenari. Font: pròpia

No és un resultat molt proper a la realitat, però en tractar-se d'un element del fons de l'escenari, serà suficient per donar un aspecte més realista.

8.4 POST PROCESSAT

Per a mostrar l'escenari com a una imatge a la pantalla, l'ordinador o l'equip on s'executa s'encarrega de renderitzar cada un dels píxels fins a formar un fotograma. Això es repeteix múltiples cops cada segon. El postprocessament és un treball extra que es fa després de renderitzar aquest fotograma. Normalment, es fa servir per aplicar algun tipus de modificació a la imatge renderitzada (antialiàsing per eliminar les vores mal definides d'alguns objectes, oclusió ambiental, modificar la tonalitat de la imatge...). Cal anar amb cura amb aquests efectes, ja que és treball extra que estem introduint a cada fotograma i pot afectar negativament al rendiment de l'aplicació.

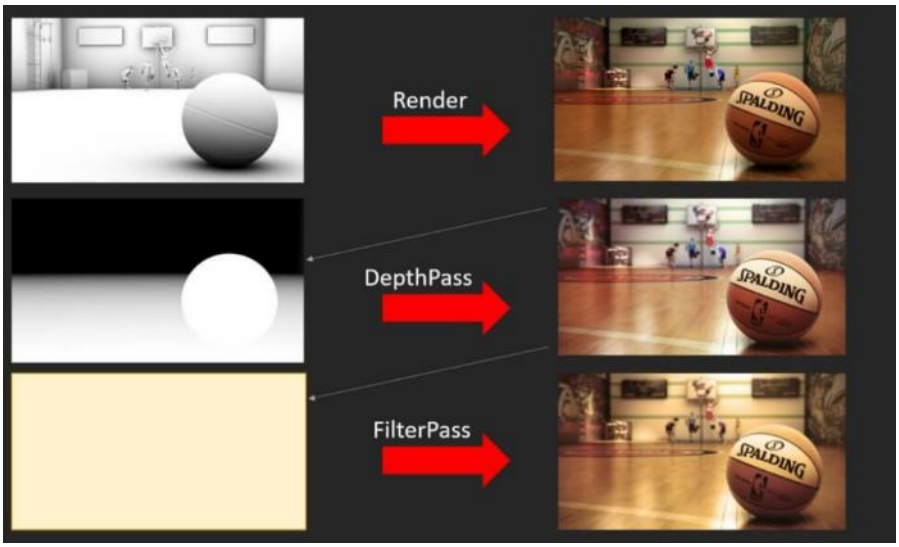

Figura 117 Funcionament del postprocessat amb exemples. Font: [68]

En aquest cas, fem servir un efecte anomenat bloom, que s'encarrega de mirar les zones o regions més brillants que tinguin un bon contrast i aplicar un desenfocament. Això permet millorar notablement el realisme d'alguns elements com per exemple l'estrella o un planeta molt calent amb roca fosa sobre la superfície.

Com fem servir llibreries del propi Three.js, caldrà importar-les.

```
import { EffectComposer  } from
'https://unpkg.com/three@0.139/examples/jsm/postprocessing/EffectComposer.js';
import { RenderPass } from
'https://unpkg.com/three@0.139/examples/jsm/postprocessing/RenderPass.js';
import { UnrealBloomPass } from
'https://unpkg.com/three@0.139/examples/jsm/postprocessing/UnrealBloomPass.js';
```

En comptes de fer servir un objecte *WebGLRenderer* com a encarregat de renderitzar els escenaris, passem a fer servir un *composer*.

A aquest li indiquem quina escena i càmera farà servir per a la tasca i a continuació creem l'efecte amb els paràmetres necessaris per aconseguir el resultat que volem.

```
this.composer = new EffectComposer( this.renderer );
this.composer.addPass(  new RenderPass( this.scene, this.camera ) );
this.composer.addPass(  new UnrealBloomPass( new THREE.Vector2( window.innerWidth,
window.innerHeight ), 2.5, 1., 0.6 ) );
```

Finalment, cal substituir la instrucció per renderitzar, per a fer servir aquest nou *composer*, de forma que substituïm aquest alinea de codi:

```
this.renderer.render( this.scene, this.camera );
```

Per aquesta:

```
this.composer.render();
```

Com podem veure a les figures, aquest efecte dona un extra de realisme a l'escena i es pot implementar molt fàcilment.

Figura 118 Planeta en formació amb una superfície molt calenta i l'efecte *bloom*. Font: pròpia

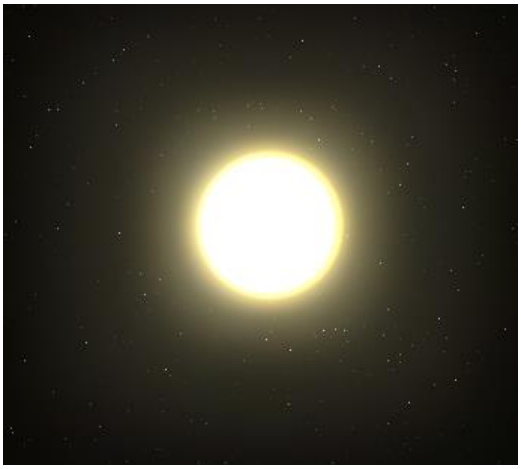

Figura 119 Estrella a l'escenari amb l'efecte *bloom*. Font: pròpia

8.5 ESCENARIS ALTERNATIUS

Per a mostrar el potencial del programa amb la base que s'ha desenvolupat, s'han creat un parell d'escenaris a més de la simulació principal on es visualitza l'evolució temporal d'un planeta com la Terra.

En el segon escenari es genera un planeta totalment cobert d'aigua, amb molt poc terreny que formen petites illes. Per a obtenir aquest resultat, nomes es necessari modificar alguns paràmetres del codi de generació de terreny per a augmentar el nivell de l'aigua. De la mateixa forma podem generar diversos tipus de planetes com al tercer escenari, on es tracta de simular l'aparença del planeta vermell. Podem veure els resultats a la Figura 120 i Figura 121.

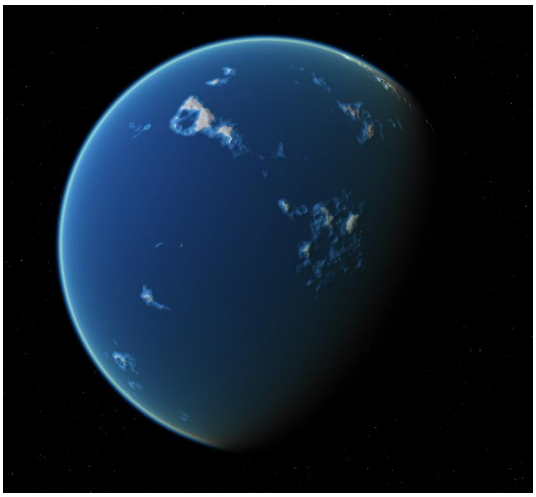

Figura 120 Segon escenari representant un planeta aquàtic. Font: pròpia

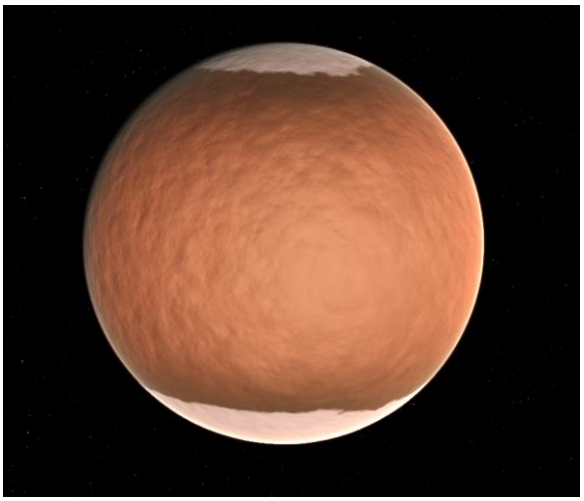

Figura 121 Tercer escenari representant un planeta semblant a Mart. Font: pròpia

CONCLUSIONS

Simular la formació de planetes i els processos que intervenen és una tasca molt complexa, encara que es simplifiqui i es duguin a terme aproximacions. Amb tot i això, s'han pogut complir els objectius proposats en aquest projecte. S'ha demostrat que es possible implementar els principals propòsits del treball en les diferents tecnologies que hi ha disponibles.

Aconseguir que el programa s'executi de forma fluida en hardware més contingut continua sent un repte, sobretot a l'hora d'actualitzar les textures generades amb els càlculs fets als diversos *shaders* que hi ha.

Lamentablement, es tracta d'una àrea en la qual només s'ha pogut veure la punta de l'iceberg, a causa de la gran complexitat del mateix i la gran càrrega de treball que requereix cobrir tots els aspectes possibles.

Pel que fa a **Three.JS**, ha demostrat ser un *framework* molt potent i ha complert amb les expectatives i les necessitats del projecte.

TREBALL FUTUR

Com a treball futur, existeixen moltes millores que es podrien realitzar al llarg del treball; la primera a mencionar podria ser el rendiment. Encara que s'ha tractat de simplificar els càlculs, el programa en general continua sent molt costós d'executar i es necessita un equip relativament modern per a poder reproduir les simulacions amb fluïdesa. Caldria dedicar temps a investigar per veure els costos computacionals de cada un dels components i tractar de treballar sobre aquests que siguin més conflictius. També sospito que gran part d'aquest problema és a la gestió de la càrrega de textures generades al *shaders* cap a la mateixa geometria de l'escenari. A més a més es podrien aplicar altres estratègies per mitigar aquests efectes com el nivell de detall (*level of detail*).

Per una altra part, es podria ser més rigorós a l'hora de simular cada un de les característiques que hem vist, així com ampliar-les i introduir més variables per poder generar resultats més precisos, realistes i de millor qualitat. Tampoc podem oblidar que, amb aquesta base que hem definit, és relativament fàcil extrapolar el simulador gràfic per a considerar incloure molts més tipus d'objectes astronòmics i variacions d'aquests. Planetes gasosos, amb anells, llunes, asteroides o estrelles, per mencionar alguns.

BIBLIOGRAFIA

[1] Patricio Gonzalez Vivo & Jen Lowe - The book of shaders
https://thebookofshaders.com/ [Consultada 13/03/2022]

[2] National Geographic - Formation of Earth
https://www.nationalgeographic.org/article/formation-earth [Consultada 28/02/2022]

[3] National Geographic - The History of Earth: How Our Planet Formed
https://www.imdb.com/title/tt1985159/ [Consultada 28/02/2022]

[4] SSERVI NASA - Widespread mixing and burial of Earth's Hadean crust by asteroid impacts
https://www.nasa.gov/ames/new-nasa-research-shows-giant-asteroids-battered-early-earth/ [Consultada 30/02/2022]

[5] Robert R. Herrick Research Associate Professor, Geophysical Institute, University of Alaska, Fairbanks. - Meteorite crater
https://www.britannica.com/science/meteorite-crater [Consultada 10/03/2022]

[6] Marco Scaioni, Vasil Yordanov, Maria Teresa Brunetti, Maria Teresa Melis,Angelo Zinzi, Zhizhong Kang & Paolo Giommi (2018) Recognition of landslides in lunar impact craters, European Journal of Remote Sensing, 51:1, 47-61
https://doi.org/10.1080/22797254.2017.1401908 [Consultada 10/03/2022]

[7] David A Roberts - Simulating worlds on the GPU
https://davidar.io/post/sim-glsl [Consultada 30/02/2022]

[8] Holger Ludvigsen - Procedural Planet in WebGL and Three.js
https://blogg.bekk.no/procedural-planet-in-webgl-and-three-js-part-2-33d99bbb2256 [Consultada 20/03/2022]

[9] Lewy Blue - Discover three.js: A Brief Introduction to Texture Mapping
https://discoverthreejs.com/book/first-steps/textures-intro/ [Consultada 21/03/2022]

[10] Ricardo Cabello - Three.js Manual: Fundamentals
https://threejs.org/manual/#en/fundamentals [Consultada 28/02/2022]

[11] Sasha Warren, University of Chicago - How the Earth and moon formed, explained
https://news.uchicago.edu/explainer/formation-earth-and-moon-explained [Consultada 01/04/2022]

[12] Steven Worley - A Cellular Texture Basis Function, pg. 291
http://www.rhythmiccanvas.com/research/papers/worley.pdf [Consultada 03/04/2022]

[13] Weisstein, Eric W. "Voronoi Diagram." From MathWorld--A Wolfram Web Resource.
https://mathworld.wolfram.com/VoronoiDiagram.html [Consultada 03/04/2022]

[14] Sanchez-Lavega, Agustin (2010). An Introduction to Planetary Atmospheres
Taylor & Francis. ISBN 978-1-4200-6732-3.

[15] Evolution Of The Atmosphere: Composition, Structure And Energy - University of Michigan
https://globalchange.umich.edu/globalchange1/current/lectures/Perry_Samson_lectures/evolution_atm/ [Consultada 13/05/2022]

[16] Astronomy 121: The Formation and Evolution of the Solar System - University of Oregon
http://abyss.uoregon.edu/~js/ast121/lectures/lec14.html [Consultada 13/05/2022]

[17] The outer planets - Laboratoy for Atmospheric and Space Physics (University of Colorado Boulder)
https://lasp.colorado.edu/outerplanets [Consultada 13/05/2022]

[18] Seki, K.; Elphic, R. C.; Hirahara, M.; Terasawa, T.; Mukai, T. (2001). "On Atmospheric Loss

of Oxygen Ions from Earth Through Magnetospheric Processes". doi:10.1126/science.1058913. PMID 11239148. S2CID 17644371.

[19] "Scientists Detected An Incoming Asteroid The Size Of A Car Last Week - Why That Matters To Us". Forbes.
https://www.forbes.com/sites/marshallshepherd/2019/06/27/scientists-detected-an-incoming-asteroid-the-size-of-a-car-last-week-why-that-matters-to-us/?sh=44d5d2564869 [Consultada 13/05/2022]

[20] Scratchapixel - Simulating the Colors of the Sky
https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/simulating-sky [Consultada 18/04/2022]

[21] Sean O'Neil - GPU gems 2 : programming techniques for high-performance graphics and general-purpose computation - Accurate Atmospheric Scattering (ISBN 0-321-33559-7)
https://developer.nvidia.com/gpugems/gpugems2/part-ii-shading-lighting-and-shadows/chapter-16-accurate-atmospheric-scattering [Consultada 18/04/2022]

[22] Alan Zucconi - Volumetric Atmospheric Scattering
https://www.alanzucconi.com/2017/10/10/atmospheric-scattering-1/ [Consultada 18/04/2022]

[23] Alan Zucconi - Rayleigh Scattering Equation
https://www.shadertoy.com/view/4tsyWs [Consultada 07/06/2022]

[24] The electromagnetic spectrum By Marcus Griffiths
https://theelectromagneticspectrumgriffithsm.weebly.com/3-which-colour-in-the-visible-light-spectrum-has-the-least-energy.html [Consultada 07/06/2022]

[25] Sebastian Lague - Coding Adventure: Atmosphere
https://www.youtube.com/c/SebastianLague [Consultada 07/06/2022]

[26] Wikipedia - History of Earth
https://en.wikipedia.org/wiki/History_of_Earth [Consultada 9/06/2022]

[27] Jordi Rodó - TEORIA DE LA TECTÒNICA DE PLAQUES
http://wiki1re01.pbworks.com/w/page/50904434/W1E02ORD16 [Consultada 10/06/2022]

[28] Tobias C. Owen of the Institute for Astronomy in Honolulu, Hawaii - What do we know about the origin of the earth's oceans?
https://www.scientificamerican.com/article/what-do-we-know-about-the/ [Consultada 12/06/2022]

[29] Earth's Early Atmosphere and Oceans
https://www.lpi.usra.edu/education/timeline/gallery/slide_17.html [Consultada 12/06/2022]

[30] Job Talle - Simulating hydraulic erosion
https://jobtalle.com/simulating_hydraulic_erosion.html [Consultada 13/06/2022]

[31] Cristina Novillo - Erosión hídrica: definición, tipos, causas y consecuencias
https://www.ecologiaverde.com/erosion-hidrica-definicion-tipos-causas-y-consecuencias-2172.html [Consultada 13/06/2022]

[32] Richard Barnes - Accelerating a fluvial incision and landscape evolution model with parallelism
https://arxiv.org/abs/1803.0297 [Consultada 13/06/2022]

[33] Wikipedia - Stream power law
https://en.wikipedia.org/wiki/Stream_power_law [Consultada 14/06/2022]

[34] ck12 - Landforms from Stream Erosion and Deposition
https://www.ck12.org/earth-science/landforms-from-stream-erosion-and-deposition/lesson/landforms-from-stream-erosion-and-deposition-hs-es/ [Consultada 14/06/2022]

[35] Grasshopper Geography - River basins of Europe
https://www.grasshoppergeography.com/ [Consultada 14/06/2022]

[36] Paul Bourke - Diffusion Limited Aggregation
http://paulbourke.net/fractals/dla/ [Consultada 14/06/2022]

[37] Wikipedia - Plate tectonics
https://en.wikipedia.org/wiki/Plate_tectonics [Consultada 17/06/2022]

[38] Joseph A. DiPietro, in Geology and Landscape Evolution (Second Edition) - Movement of Tectonic Plates
https://www.sciencedirect.com/science/article/pii/B9780128111918000051 [Consultada 17/06/2022]

[39] National Geographic - Plate Tectonics
https://www.nationalgeographic.com/science/article/plate-tectonics [Consultada 17/06/2022]

[40] Irene Martínez Pérez - Les plaques tectòniques
http://www.xtec.cat/~imartin6/1/hotpot/tectonica/tectonica3.htm [Consultada 17/06/2022]

[41] GitHub Home
https://github.com/home [Consultada 17/06/2022]

[42] NASA - Mars 2020 Entry Descent Landing
https://eyes.nasa.gov/apps/mars2020/#/home [Consultada 17/06/2022]

[43] SpaceX - ISS Docking Simulator
https://iss-sim.spacex.com/ [Consultada 17/06/2022]

[44] Steven Wittens - Making Worlds 1 - Of Spheres and Cubes
https://acko.net/blog/making-worlds-1-of-spheres-and-cubes/ [Consultada 17/06/2022]

[45] BERNAT ORELLANA BECH - UPC: Interacció i Disseny d'Interfícies - Procés de visualització

[46] ThreeJS – MeshStandardMaterial
https://threejs.org/docs/#api/en/materials/MeshStandardMaterial [Consultada 18/06/2022]

[47] wikiwand– Atmosphere
https://www.wikiwand.com/en/Atmosphere [Consultada 18/06/2022]

[48] Tomoyuki Nishita - Display of The Earth Taking into Account Atmospheric Scattering
https://www.researchgate.net/publication/2933032_Display_of_The_Earth_Taking_into_Account_Atmospheric_Scattering [Consultada 18/06/2022]

[49] N Hoffman - Rendering Outdoor Scattering in Real Time
http://developer.amd.com/wordpress/media/2012/10/GDC_02_HoffmanPreetham.pdf [Consultada 18/06/2022]

[50] NASA Michael Summers New Horizons Co-Investigator - Science Shorts: How Big Is Pluto’s Atmosphere?
http://pluto.jhuapl.edu/News-Center/Science-Shorts.php?page=ScienceShorts_02_24_2015 [Consultada 19/06/2022]

[51] Wikipedia - Earth
https://en.wikipedia.org/wiki/Earth [Consultada 19/06/2022]

[52] Wikipedia - Scale height
https://en.wikipedia.org/wiki/Scale_height [Consultada 19/06/2022]

[53] NASA - NASA Captures "EPIC" Earth Image
https://www.nasa.gov/image-feature/nasa-captures-epic-earth-image [Consultada 19/06/2022]

[54] Redox state of Earth's magma ocean and its Venus-like early atmosphere, Sossi, P.A., Burnham, A. D., Badro, J., Lanzirotti, A., Newville, M. & O'Neill, H.St.C., Science Advances. doi: 10.1126/sciadv.abd1387
https://nccr-planets.ch/blog/2020/11/26/new-insights-into-earths-primeval-atmosphere/ [Consultada 20/06/2022]

[55] Geoff's Climate Cookbook
https://web.archive.org/web/20130619132254/http://jc.tech-galaxy.com/bricka/climate_cookbook.html [Consultada 21/06/2022]

[56] National Ocean Service - The Coriolis Effect
https://oceanservice.noaa.gov/education/tutorial_currents/04currents1.html [Consultada 21/06/2022]

[57] Testbook - Intertropical Convergence Zone (ITCZ), a low-pressure zone, is located
https://www.globe.gov/explore-science/scientists-blog/archived-posts/sciblog/index.html_p=1661.html [Consultada 21/06/2022]

[58] Wikipedia – Atmospheric pressure
https://en.wikipedia.org/wiki/Atmospheric_pressure [Consultada 22/06/2022]

[59] PhysicalGeography | FUNDAMENTALS eBOOK - CHAPTER 7: Introduction to the Atmosphere - Global Scale Circulation of the Atmosphere
http://www.physicalgeography.net/fundamentals/7p.html [Consultada 22/06/2022]

[60] Wikipedia – Gaussian blur
https://en.wikipedia.org/wiki/Gaussian_blur [Consultada 23/06/2022]

[61] Nullschool - Wind + Temperature @ Surface on Earth (Data from: GFS / NCEP / US National Weather Service)
https://earth.nullschool.net/#2021/02/25/1800Z/wind/surface/level/overlay=temp/ [Consultada 23/06/2022]

[62] CENTURY Soil Organic Matter Model Environment - Great Plains System Research Unit Technical Report No. 4 USDA-ARS Fort Collins Colorado
https://www2.nrel.colostate.edu/projects/century/MANUAL/html_manual/man96.html#FIG3-8B [Consultada 24/06/2022]

[63] Wikipedia - Biome
https://en.wikipedia.org/wiki/Biome [Consultada 24/06/2022]

[64] Simulation of water surface using current consumer-level graphics hardware
https://www.researchgate.net/figure/1D-Perlin-noise-noise-from-a-to-f-with-halved-amplitude-and-doubled-frequency-to-the_fig8_327042430 [Consultada 25/06/2022]

[65] Leiden University - Biological Content Generation: Evolving Game Terrains Through Living Organisms
https://www.researchgate.net/figure/Perlin-noise-pattern-represented-as-greyscale-image-left-and-the-resulting-terrain_fig1_274384740 [Consultada 25/06/2022]

[66] Wikipedia - Perlin noise
https://en.wikipedia.org/wiki/Perlin_noise [Consultada 24/06/2022]

[67] Omar Bradley - Star Colors: Why They Differ and What We Can Learn From Them
https://www.color-meanings.com/star-colors [Consultada 24/06/2022]

[68] Red Stapleer – Three.js Postprocessing Effect Tutorial
https://redstapler.co/threejs-postprocessing-effect-tutorial/ [Consultada 25/06/2022]