Màster en Computació

Departament de Llenguatges i Sistemes Informàtics

Universitat Politècnica de Catalunya

Il·luminació Global per Ciutats

Tesi de Màster

Ferran Roure Garcia

Tutors:

Dr. Gustavo Patow

Dr. Gonzalo Besuievsky

Setembre 2011

ABSTRACT

El càlcul de la il·luminació global per grans models urbans és encara un problema obert. El cost computacional d'aquest càlcul fa que que trobar una solució a aquest problema no sigui una tasca fàcil.

En aquesta tesi de màster hem proposat crear un sistema que calculi la il·luminació global d'una ciutat aprofitant la naturalesa procedural dels nostres edificis, utilitzant mètodes de radiositat jeràrquica a l'hora de distribuir la llum del sol entre els elements que no estan directament il·luminats.

AGRAÏMENTS

M'agradaria agrair als meus dos tutors, Gus i Gonzalo, el suport que m'han donat durant tot el procés d'elaboració d'aquesta tesi, als meus companys del despatx i de fora el despatx, per suportar totes les meves preguntes i a la meva família per recolzar-me durant tot aquest temps.

Moltes gràcies a tots.

Índex

Índex de figures

1. Introducció

Avui en dia el modelatge de ciutats és un problema obert pel qual no existeixen solucions estàndards ni de codi lliure. Aquest és un problema important per a indústries com el cinema, la realitat virtual o els videojocs. Dins d'aquest problema es poden presentar sub-problemes interessants, com ara el modelatge geomètric dels edificis, la distribució dels mateixos dins un context urbà prèviament planificat, o la possibilitat de representar aquests models amb una alta qualitat visual. Aquesta última opció és el tema central d'aquesta tesi de màster.

1.1 Motivació

Durant els últims anys, l'interès pel modelatge procedural, ja sigui d'edificis, ciutats o de qualsevol altra cosa, ha anat creixent de manera significativa. Els investigadors s'han adonat que aquestes tècniques permeten manipular grans volums de dades d'una manera efectiva. Actualment però, les línies de recerca basades en tècniques procedurals centren el seu interès en el modelatge geomètric, deixant de banda l'aspecte visual dels resultats. Per aquest motiu, la nostra intenció és començar una nova línia de recerca centrada en aconseguir visualitzar grans volums de dades, en el nostre cas, una ciutat, utilitzant tècniques procedurals tan per a la creació de geometria com per el càlcul de la il·luminació realista. Més concretament, la idea és fer el càlcul de la il·luminació global d'una ciutat de mida mitjana aprofitant la naturalesa procedural dels edificis que rebem com a input.

1.2 Objectius

L'objectiu principal del projecte és aconseguir un sistema que calculi la il·luminació global d'una ciutat, tenint en compte l'energia directa del sol i l'intercanvi d'energia entre edificis, fet produït per la reflexió difosa dels rajos incidents del sol sobre les façanes dels edificis. Si bé no és un objectiu primari, també ens hem proposat de crear l'aplicació de forma que sigui independent de qualsevol programari de modelat o de visualització. La Figura 1.1 representa els passos que seguirà el nostre algorisme.

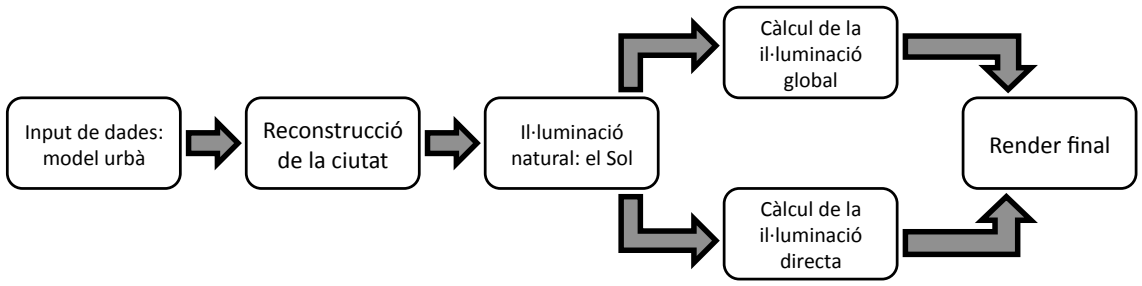


Figura 1.1: Diagrama de seqüència del nostre projecte.

D'entrada rebrem un model urbà com a input de dades. Reconstruirem el model i el prepararem per a la visualització i també per a el càlcul de la il·luminació global. Aquesta il·luminació serà calculada en dues fases: la il·luminació de baixa frequència o il·luminació global i la il·luminació d'alta freqüència o llum directa. La correcta combinació d'aquest dos càlculs donarà lloc a la visualització final, aplicant aquests càlculs d'il·luminació sobre la geometria del l'escena.

1.3 Planificació

La planificació del projecte es divedeix en dues grans parts: el plantejament incial i la implementació. Durant l'inici del segon semestre acadèmic del curs 2010-2011 es va

pantejar l'objectiu del projecte, es va fer una cerca dels treballs existents sobre el tema i es van proposar solucions. L'altre part important ha estat la implementació d'aquestes solucions. Com es pot veure a la Figura 1.2, la part de reconstrucció de la ciutat ha estat la que ha requerit més temps i la que ens ha donat més problemes, tot i no ser el tema central del projecte. La part d'il·luminació s'ha desenvolupat tal i com estava previst.

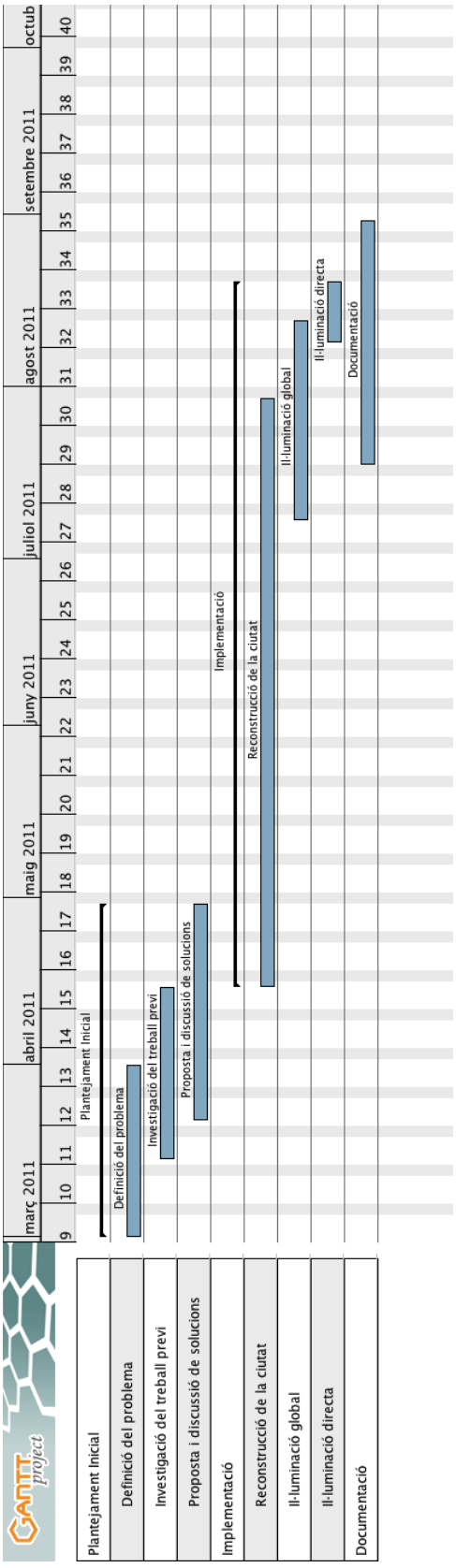


Figura 1.2: Diagrama de Gantt del projecte.

2. Treball Previ

2.1 Modelatge Urbà

Una de les primeres tècniques procedurals de generació de ciutats en 3D va ser introduïda per Parish i Müller [4] presentant una pipeline pel modelatge urbà utilitzant L-Systems [6] per la creació de carrers i edificis (veure Figura 2.1). Sun et al. [8] van presentar també una aproximació similar per generar xarxes de carrers, basada en el sistema de substitució de patrons. Aquests mètodes van ser estesos per la creació d'edificis procedurals [9] [3] aconseguint crear façanes que imiten l'estructura real dels elements arquitectònics que les componen.

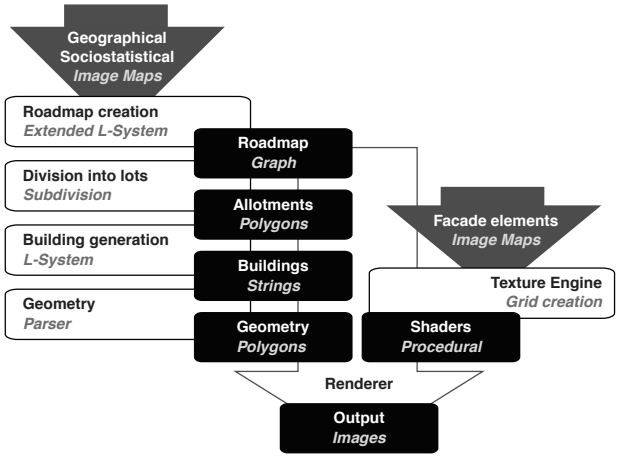


Figura 2.1: Pipeline de modelatge urbà creat per Parish i Müller [4].

Els L-Systems [6] o Sistemes de Lindenmayer, són la base del modelatge procedural que s'utilitza als articles de Müller et. al. [3] per a la creació d'edificis (veure Figura 2.2). Aquests sistemes es basen en regles que s'auto-alimenten per generar més regles que, al final, acaben definint una estructura coherent en forma d'arbre. El funcionament bàsic dels L-Systems consisteix en substituir cadenes de caràcters per altres de més complexes de manera iterativa, tantes vegades com nivells defineixi l'usuari. Ho podem veure com:

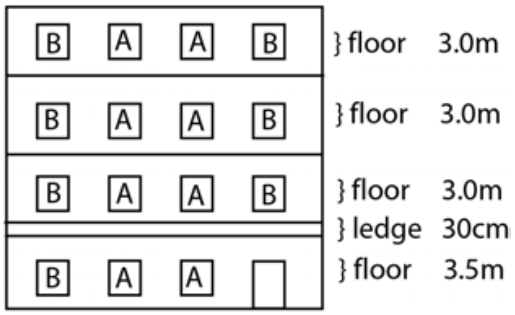
$$G = \{V, S, w, P\}$$

on

- **V** és un conjunt de símbols que conté elements que poden ser substituïts (variables).
- **S** és un conjunt de símbols que conté elements que es mantenen fixes (constants).
- w és una cadena de símbols de **V** que defineixen l'estat inicial del sistema (inici o axioma).
- **P** és un conjunt de regles o produccions que defineixen la manera com les variables poden ser substituïdes per combinacions de constants i altres variables. Una sola producció està formada per dues cadenes: el predecessor i el successor.

Les regles gramaticals dels L-Systems s'apliquen de manera iterativa a partir de l'estat inicial.

Seguint aquestes directrius, Müller et. al. [3] van desenvolupar una sintaxi que permet crear edificis de manera semi-automàtica proporcionant a l'usuari un sistema ràpid per la creació de grans conjunts urbans (Figura 2.2). Tot i això, aquest procés es torna feixuc i repetitiu degut a la molèstia d'haver de crear llargues regles sintàctiques per a poder obtenir edificis amb suficient nivell de detall.



1: fac $\rightsquigarrow$ Subdiv("Y",3.5,0.3,3,3,3){ floor | ledge | floor | floor | floor }

Figura 2.2: Exemple d'una regla per a la creació d'una façana segons l'article de Müller et. al [3]. La façana es subdivideix en l'eix de les Y i se n'obté la planta baixa i tres pisos.

Patow [5], per altra banda, proporciona un sistema amigable, amb interfície gràfica i associat a l'skylineEngine [7], per la creació de ciutats procedurals, representant aquestes ciutats en un graf visual de regles (veure Figura 2.3).

Figura 2.3: A l'esquerra es mostra un exemple de la modificació d'un edifici mitjançant el mètode proposat per Patow [5] basat en un graf visual. A la dreta es mostra els resultats obtinguts amb el sistema skylineEngine [7].

2.2 Radiositat Jeràrquica

El mètode de radiositat jeràrquica que hem seguit pel nostre projecte va ser introduït per Hanrahan [2] i recollit després al llibre de Cohen i Wallace [1]. Aquest mètode consisteix, en línies generals, en distribuir l'energia procedent de les fonts de llum cap a tots els elements de l'escena, aproximant la il·luminació global i substituint el terme ambient que hi havia fins aleshores. El concepte és, si més no, intuïtiu: compartir part de l'energia rebuda per uns elements de l'escena cap a uns altres seguint uns paràmetres determinats, amb diferents nivells d'interacció en funció de la magnitud d'energia que intercanvien les superfícies. El problema d'aquest plantejament és, sobretot, el cost computacional que té. Per acostar la simulació cap a un grau realisme adequat cal idear un sistema que aconsegueixi fer creïbles els resultats, però amb uns costos acceptables per les màquines actuals.

La millor opció, segons els autors, és utilitzar un tipus d'estructura jeràrquica que permeti descartar grans volums de càlculs de manera ràpida. En el seu cas, utilitzen un *quadtree* (Figura 2.4) per representar cadascun dels elements que intervé en l'intercanvi d'energia.

L'algorisme consta bàsicament de dos passos: compartir energia i distribuir energia (veure Figura 2.5). El pas de compartir energia comença establint una sèrie d'enllaços entre elements de diferents objectes. Donats dos elements de superfície d'una escena (patches), l'algorisme estableix una relació (link) entre ells que servirà per intercanviar energia. La part interessant del mètode és decidir si un patch és vàlid per intercanviar energia o si és necessari dividir-lo i explorar els seus fills per tal d'obtenir un resultat amb més resolució. Aquest test té en compte si l'àrea dels patches i el Factor de Forma entre ells són més grans que uns certs llindars. En cas afirmatiu, un dels patches, habitualment el d'àrea i Factor de Forma més gran, serà dividit. Es repeteix el procés fins que els patches tenen una àrea per sota d'un determinat llindar.

El Factor de Forma és un valor que ajuda a decidir si la possible aportació d'energia d'un

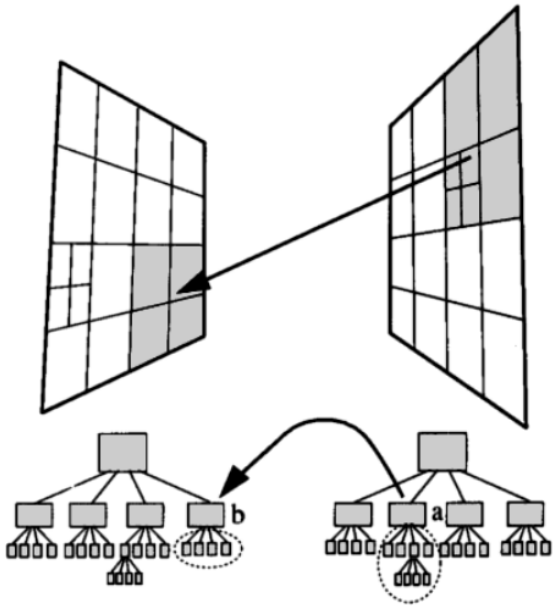


Figura 2.4: Representació conceptual i la seva equivalència a l'estructura de dades del llibre de Cohen i Wallace [1].

```
SolveSystem(){
  Until Converged
    for ( all sufraces p ) GatherRad( p );
    for ( all sufraces p ) PushPullRad( p, 0.0 );
}
```

Figura 2.5: Pseudocodi de l'algorisme de radiositat jeràrquica. Aquest mètode s'executa iterativament fins que l'aportació d'energia és inapreciable.

patch a l'altre és suficientment significativa per dur-la a terme. És un factor geomètric que es defineix com la proporció de la radiació que pot sortir del patch emissor per arribar al patch receptor. Tot i que hi ha moltes maneres de calcular-lo, els autors aproximen el seu valor utilitzant la següent fórmula:

$$F_{pq} \approx \frac{cos\theta}{\pi} w_q$$

Aquesta estimació es calcula des dels centres de les àrees representades pels patches p i q. El factor w_q és l'angle sòlid sota el qual es veu una esfera (o un disc) envolupant de l'àrea del node q (veure Figura 2.6).

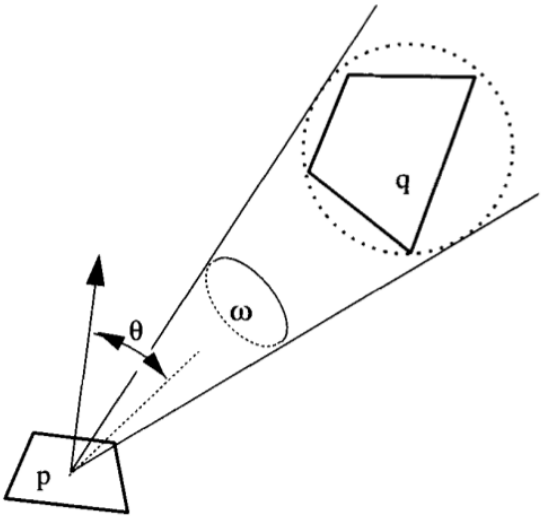


Figura 2.6: Geometria simple per aproximar el Factor de Forma.

El mètode que s'encarrega d'avaluar els patches i crear els links és un mètode anomenat *oracle* (Figura 2.7). La tasca d'aquest mètode és fer una estimació de l'error que hi pot haver enllaçant dos paches.

Un cop construïda l'estructura de dades amb els enllaços pertinents, comença el procés de radiositat jeràrquica (Figura 2.5). El primer mètode que hi intervé (Figura 2.8) és el que s'encarrega de compartir l'energia a través dels enllaços creats prèviament. Aquesta funció té en compte un seguit de paràmetres com ara el coeficient de reflexió del material, el Factor de Forma entre dos elements i, per descomptat, la radiositat que ha de passar d'un patch a un altre.

L'últim pas de l'algorisme consisteix en redistribuir l'energia rebuda per tots els nodes del *quadtree*. Aquesta tasca és essencial sobretot de cares a les següents iteracions del mètode. La funció *PushPull* (Figura 2.9) distribueix l'energia rebuda (*Bg*) per un node cap els seus fills i torna el resultat ponderat, dels fills cap al pare, en relació l'àrea dels

```
float Oracle( Quadnode p, Quadnode q, float Fe ){
  if ( p.area < Ae and q.area < Ae )
    return ( FALSE );
  if ( EstimateFormFactor ( p, q ) < Fe )
    return ( FALSE );
  else
    return ( TRUE );
}
```

Figura 2.7: Pseudocodi del mètode *oracle*. '*Fe*' és el valor del Factor de Forma mínim i '*Ae*' és l'àrea mínima, a partir dels quals es considerarà si cal dividir el node.

```
GatherRad ( Quadnode p ){
  Quadnode q; Link L;
  p.Bg = 0;
  for ( each gathering link L of p ) //gather energy across the link
    p.Bg += p.ir * L.Fpq * L.q.Bs;
  for ( each child node r of p )
    GatherRad( r );
}
```

Figura 2.8: Pseudocodi del mètode *GatherRad*. '*Bg*' és la radiositat d'entrada d'un node. '*Fpq*' és el factor de forma entre els nodes candidats. '*Bs*' és la radiositat de sortida d'un node.

fills. Les fulles del *quadtree* reben la suma de la radiositat rebuda pels pares. Aquesta radiositat és retornada cap als pares en forma de '*potència/àrea*'. Per exemple, si quatre fills tenen tots la mateixa radiositat, el seu pare tindrà també la mateixa radiositat, ja que aquesta ve ponderada per l'àrea dels seus fills.

```
PushPull ( Quadnode p, float Bdown ){
  float Bup, Btmp;
  if ( p.children == NULL )   // p és una fulla
    Bup = p.Bg + Bdown;
  else
    Bup = 0;
    for ( each child node r of p )
      Btmp = PushPull( r, p.Bg + Bdown );
      Bup += Btmp * ( r.area / p.area );

  p.Bs = Bup;
  return Bup;
}
```

Figura 2.9: Pseudocodi del mètode *PushPull*.

Aplicant aquestes funcions en diferents passades, l'energia es distribueix entre els diferents elements de l'escena, simulant el comportament de la llum en termes d'il·luminació global.

3. Reconstrucció de la ciutat

Degut a la decisió de desvincular aquest projecte de qualsevol programa de modelat, ens va fer assumir un nou repte, previ a l'objectiu central de la tesi, que és el de la reconstrucció dels models geomètrics. És important notar que diem "reconstrucció" perquè els models no són creats pel nostre algorisme, sinó que ja venen definits mitjançant un seguit de fitxers que ja explicarem més endavant.
Així doncs, en aquest capítol es podrà veure detalladament tot el procés de lectura, reconstrucció i pintat de la geometria sense il·luminació.

3.1 L'arbre de nodes

L'estructura de dades escollida per emmagatzemar els edificis és un arbre n-ari on cada node representa un element jeràrquic de l'edifici i on només els nodes fulla contenen la geometria que es pintarà. Així doncs, podem dir que cada edifici parteix d'una arrel de la qual en pengen les parets i el sostre, i d'aquests en pengen els pisos i així successivament fins a arribar als elements més petits. Aquesta estructura, a més, ens permet implementar de manera més còmode la part de radiositat jeràrquica que generarà la il·luminació global de la ciutat (veure Capítol 4).
La Figura 3.1 mostra la correspondència entre l'arbre de nodes i els elements d'un edifici.

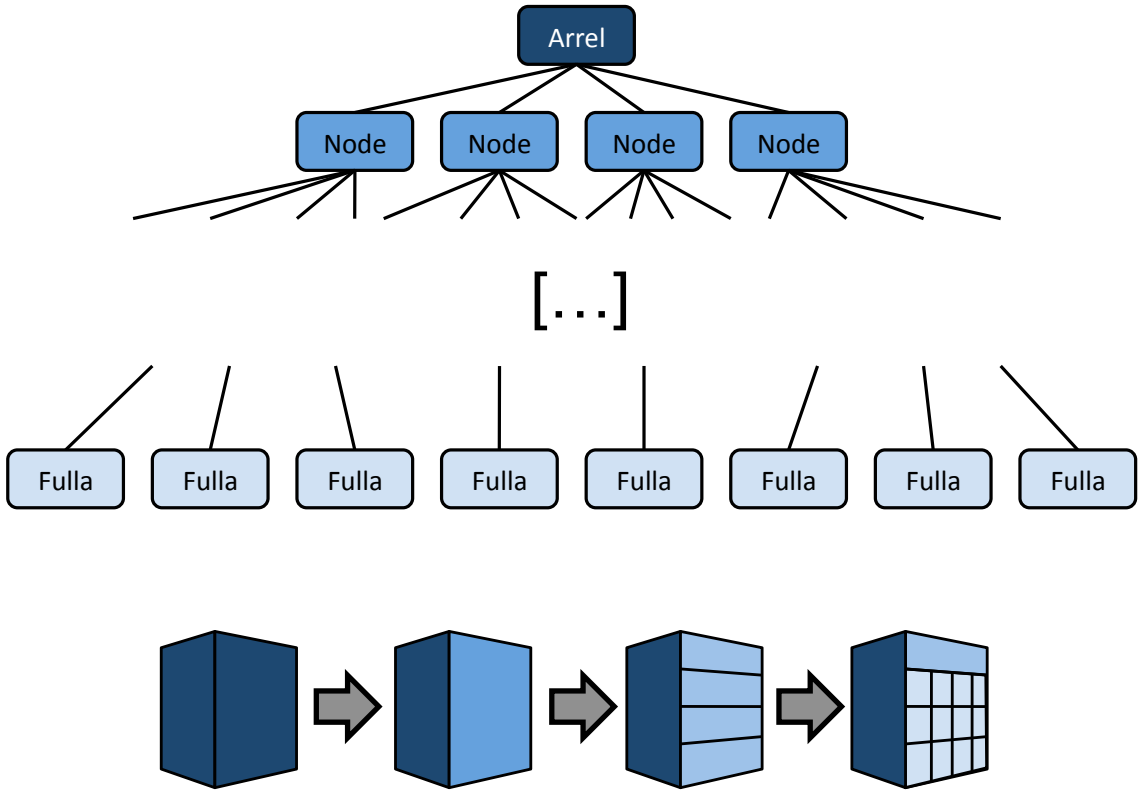


Figura 3.1: Representació de l'arbre de nodes d'un edifici i la seva correspondència amb la geometria.

3.2 El format d'entrada: productTree

El primer pas a fer és carregar la geometria que necessitarem pintar. Aquests models són creats amb l'**skylineEngine** [7] i posteriorment exportats en format XML. Actualment, l'**skylineEngine** crea els edificis generant un graf de regles procedurals que defineixen totes les parts dels edificis. Per tal de poder implementar la radiositat jeràrquica, ens cal un arbre de nodes molt ben definit. El primer problema que es planteja és passar del graf de regles a un arbre de nodes.

Per aquesta tasca, primer cal identificar cadascun dels edificis que compon l'escena per tal d'explorar-lo per separat i generar-ne l'arbre corresponent. El recorregut d'aquest sub-graf genera un arbre on el node arrel representa l'edifici complet, els seus fills representen les

façanes i així successivament fins a tenir totes les divisions que estaven marcades en el graf de regles. Després es graven les dades en un fitxer *.pT, juntament amb una llista dels assets (geometria final) que necessita, i es segueix amb el següent edifici del graf. Quan s'han explorat tots els arbres, es genera un altre fitxer *.citypT que conté les referències a tots els edificis, juntament amb una referència al terra de l'escena.
Aquest conjunt de fitxers en format XML, que hem anomenat *productTree*, són l'input del nostre projecte.

3.2.1 El fitxer *.pT

El fitxer productTree recull tota la informació d'un edifici concret. Entre els tags *buildingHierarchy* es defineix l'arbre de nodes i els assets (elements geomètrics prefabricats com finestres, portes, etc.) vinculats a l'edifici.

```
<?xml version="1.0" ?>
  <buildingHierarchy>
    <!-- Arbre de nodes que defineix l'edifici -->

  <insertList>
    <!-- Llista d'assets -->
  </insertList>
</buildingHierarchy>
```

Figura 3.2: Capçalera d'un fitxer *.pT on es defineix un edifici.

3.3 Els nodes

Com hem pogut veure a la Figura 3.1, l'estructura de dades que fem servir està composta d'elements de tipus "node". Hem creat una classe Node que conté tant la informació conceptual de l'estructura de dades (referències al seu pare i als seus fills) com la informació

geomètrica (primitiva que defineix la posició física del node i referències als possibles assets vinculats). Posarem més d'èmfasi en la descripció d'aquesta classe perquè creiem que és l'element més important de tots, ja que és la base de totes les operacions posteriors.

La Figura 3.3 mostra els principals elements de la classe Node. Cada node consta de:

- Un punter a un node pare.
- Una llista de punters a nodes fill.
- Un punter a un primitiva geomètrica (es fa servir de referència per posicionar els assets).
- Una llista de punters a assets (geometria final).

```
Class Node{
  Node *father;
  vector<Node*> children;
  Primitiva *prim;
  vector<Object*> assets;
}
```

Figura 3.3: Elements de la classe Node.

Aquest elements són genèrics i per tant, en alguns casos es feran servir i en altres no. Per exemple, el node arrel de l'arbre no té *father*, i els nodes fulla no tenen *children* però sí tenen *assets*. A la Figura 3.4 podem veure el format XML d'un node, llegit del fitxer .*pT.

```
[...]
<node>
  <prim descr="prim #0 of CompSelector2">
    <point x="-2058.00" y="0.00" z="-5989.99"/>
    <point x="-2058.00" y="1199.99" z="-5989.99"/>
    <point x="-2058.00" y="1199.99" z="-7957.99"/>
    <point x="-2058.00" y="0.00" z="-7957.99"/>
  </prim>
  <children>
    <node>
      <prim descr="prim #0 of Subdiv3">
        <point x="-2058.00" y="1200.00" z="-5989.99"/>
        <point x="-2058.00" y="1199.99" z="-7957.99"/>
        <point x="-2058.00" y="0.00" z="-7957.99"/>
        <point x="-2058.00" y="0.00" z="-5989.99"/>
      </prim>
      <insertLabel>
        Insert4
      </insertLabel>
    </node>
  </children>
</node>
[...]
```

Figura 3.4: Codi XML del fitxer *.pT que representa un node amb tots els seus elements.

3.4 Els assets

Per tal de dotar els edificis d'un aspecte visual realista, no en fem prou amb representar les parets mitjançant simples primitives. Per això fem servir models geomètrics prèviament creats per un artista, anomenats "assets", que són molt més fidedignes a l'aspecte arquitectònic d'un edifici. Aquests "assets"són elements separats (portes, finestres,...) que es

combinen i es repeteixen per a crear l'aspecte final de l'edifici. A la Figura 3.4 podem veure com el node fulla té una etiqueta que posa "Insert 4". Aquesta etiqueta fa referència a l'asset del mateix nom, que ve detallat al final de fitxer *.pT, entre els tags *insertList*.

```
<insertList>
  <insert id="Insert4">
    <param name="asset" value="downwindowsCornerBuilding.obj"/>
    <param name="sx" value="1"/>
    <param name="sy" value="1"/>
    <param name="sz" value="300"/>
    <param name="rx" value="0"/>
    <param name="ry" value="0"/>
    <param name="rz" value="0"/>
    <param name="tx" value="0"/>
    <param name="ty" value="0"/>
    <param name="tz" value="-200"/>
    <param name="relx" value="on"/>
    <param name="rely" value="on"/>
    <param name="relz" value="off"/>
  </insert>
  [...]
</insertList>
```

Figura 3.5: Exemple de la lista dels assets al fitxer *.pT

Un asset no és només una referència al fitxer que conté la geometria de l'objecte, sinó és també un conjunt de paràmetres que ens serveixen per ajustar aquests objectes. Sovint, en el procés de modelat, l'artista agafa un determinat objecte (entenem per objecte la figura geomètrica que conté vèrtexs, cares, etc, i que representa una determinada forma arquitectònica) i el modifica d'acord amb les seves idees. És possible que un objecte sigui deformat en algun dels eixos per tal de poder-lo encabir en una posició determinada. Per aquesta raó nosaltres no en tenim prou en saber quin objecte s'ha d'inserir a l'escena, sinó que també necessitem els paràmetres que ajusten aquest objecte segons el criteri de

l'artista. Els paràmetres que es poden veure a la Figura 3.5 ens donen tota la informació d'escalat (sx,sy,sz), rotació (rx,ry,rz) i translació (tx,ty,tz) necessaris per posar l'objecte on toca. Els paràmetres *rel** ens indiquen quins valors s'han d'agafar com a absoluts i quins com a relatius.

3.5 Posicionament de la geometria

Com hem dit abans, cada node conté una primitiva que és la seva representació geomètrica a l'escena. Aquestes primitives, casi sempre quads, ja representen l'estructura bàsica d'un edifici. Aquests quads són únics per cada node (això és important per poder calcular la radiositat jeràrquica que explicarem al Capítol 4), però els assets associats a cada node són només punters a objectes. Ho hem fet així per tal de no tenir elements redundats, ja que un mateix asset es pot fer servir per diferents nodes (totes les finestres d'una façana acostumen a ser iguals). Per aquest motiu només es crea una instància per cada objecte definit al fitxer *productTree* i es va pintant a diferents llocs segons ho indica la primitiva del node al qual pertany.

Aquest procés comença en el moment en el que carreguem l'objecte a memòria. Primer de tot cal posar l'objecte en una posició i orientació conegudes, ja que no tenim cap garantia que l'objecte estigui en una situació determinada. Per això apliquem a cadascun dels vèrtexs de l'objecte, una translació a l'origen de coordenades de l'escena i una rotació segons els paràmetres que ens indica l'asset. D'aquesta manera evitem possibles deformacions en el moment de fer les transformacions geomètriques necessàries per posicionar cada asset sobre la seva primitiva de referència. Aquest procés només es fa una vegada per cada asset.

Diferent d'això és el procés de pintat dels assets. Aquest procés no modifica el valor dels vèrtexs sinó que només aplica a cada asset un seguit de transformacions geomètriques que situen l'objecte a la posició marcada per la primitiva del node al que pertany. La Figura 3.6 mostra tot el procés de posicionament de la geometria segons les primitives de l'edifici i a la Figura 3.7 es pot veure el resultat d'aquest procés.

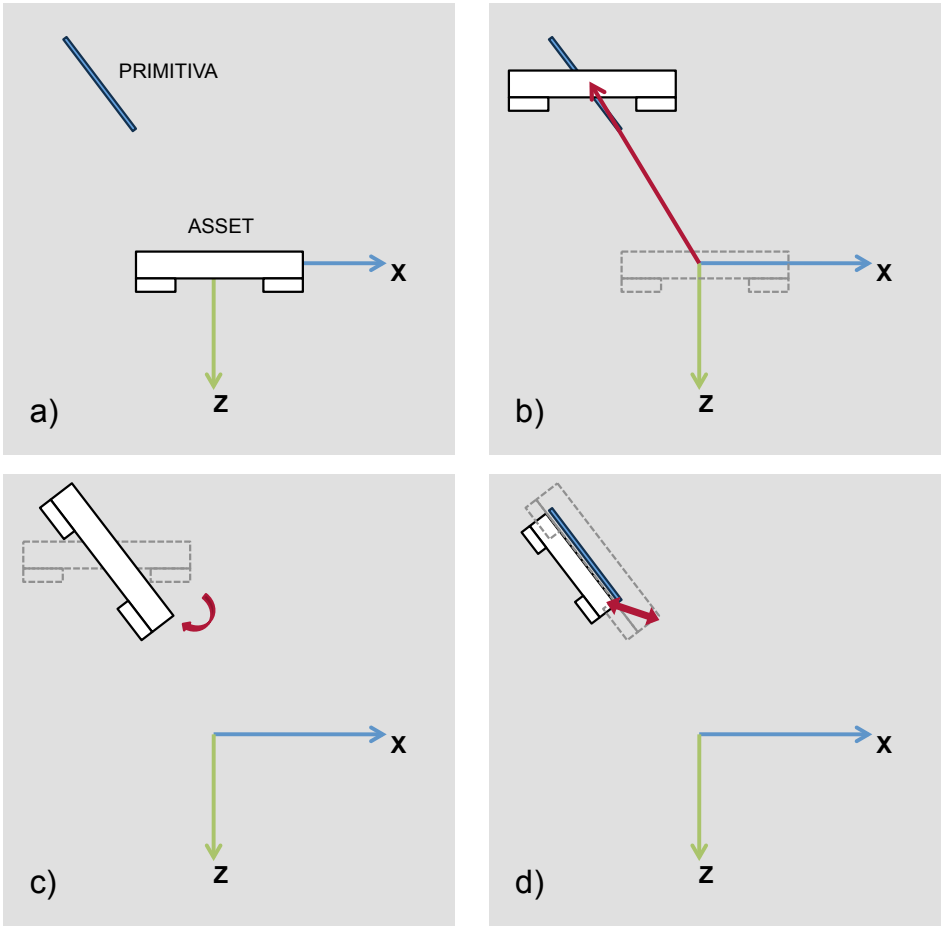


Figura 3.6: a) L'asset sempre estarà col·locat en una posició arbitrària. b) Traslladem l'asset fins al la posició de la primitiva. c) Rotem l'asset en funció de l'angle entre la Normal de la primitiva i la Normal de l'asset. d) Ajustem l'asset a la seva posició final, escalant segons les proporcions de la primitiva.

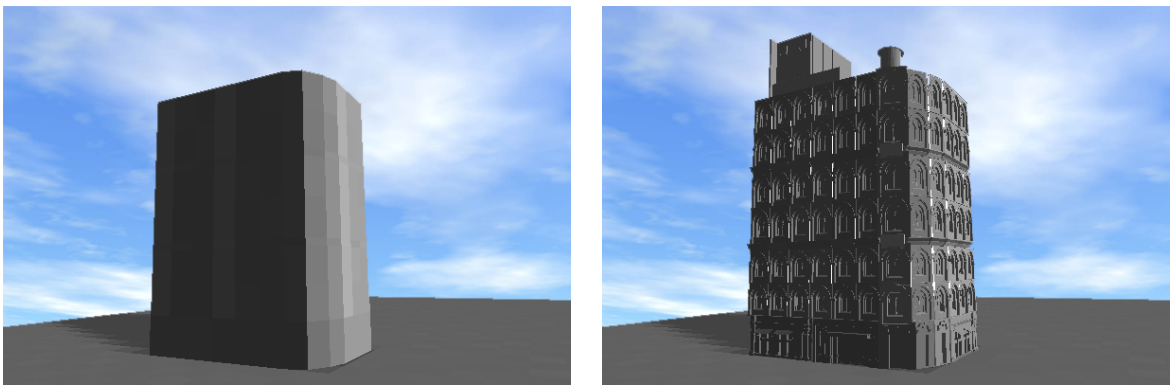


Figura 3.7: Exemple d'un edifici pintat només amb les primitives (dreta) i del mateix edifici afegint-hi la geometria final dels assets (esquerra). Per generar aquestes imatges només s'ha tingut en compte la il·luminació directa.

4. Il·luminació

En aquest capítol veurem en detall tot el sistema d'il·luminació global per ciutats, que és el tema central d'aquest projecte. Per tal d'aconseguir uns resultats realistes no en fem prou en simular la il·luminació mitjançant el sistema que ens proporciona OpenGL. Aquest sistema distribueix equitativament la il·luminació ambient per tota l'escena sense tenir en compte els elements que, sens dubte, intervenen en la distribució d'aquesta energia. Per això s'han de fer servir altres mètodes més precisos. Aprofitant la naturalesa procedural de les nostres escenes, hem decidit implementar un mètode de radiositat jeràrquica que ens permetrà redistribuir l'energia provinent de la font de llum, el sol en el nostre cas, cap a tota l'escena, a partir de les reflexions dels raigs solars. Tot aquest càlcul es fa en temps de pre-procés ja que el sol i la geometria són fixes i no cal re-calcular la radiositat.

4.1 Il·luminació global per ciutats

La idea general del projecte és aconseguir un sistema que sigui capaç de calcular la il·luminació global d'una ciutat real. Aquesta idea, a priori, es planteja poc factible a causa de la gran quantitat de dades a tractar. Per aquest raó és necessari idear mètodes que aproximin el resultat, sense deixar de banda la qualitat visual, però reduint el temps de càlcul fins a nivells acceptables.

La tècnica que hem escollit nosaltres és la **Radiositat Jeràrquica**. Donada la naturalesa procedural de les nostres escenes, aquest mètode és ideal per aconseguir el nostre

propòsit. La raó principal per escollir aquest mètode és la similitud existent entre els dos sistemes. La radiositat jeràrquica treballa amb un arbre de patches i el modelatge procedural genera un arbre de primitives (nodes). La radiositat jeràrquica genera un arbre subdividint 'sobre la marxa' cada quad en quatre parts fins un àrea mínima. En canvi, el modelatge procedural subdivideix cada façana fins la mida dels assets (veure Figura 4.2). Parlant d'edificis, nosaltres postulem que aquest nivell, el de l'asset, és suficient per simular la il·luminació indirecta. Com es pot observar a la Figura 4.1, la diferència entre la component de lluminositat del dos punts mostrejats de l'imatge és molt petita.

Figura 4.1: Foto d'una façana d'un edifici. Es pot veure com a la part on no hi ha il·luminació directa, la diferència de lluminositat entre els dos punts mostrejats, que estan a una distància considerable l'un de l'altre, no és molt gran (component L del HSL).

Respecte del procés de divisió en si, la radiositat jeràrquica divideix cada patch en quatre, generant un quadtree, mentre que el modelatge procedural genera un arbre n-ari (veure Figura 3.1). A més, la forma geomètrica dels nodes pot ser molt diferent (un

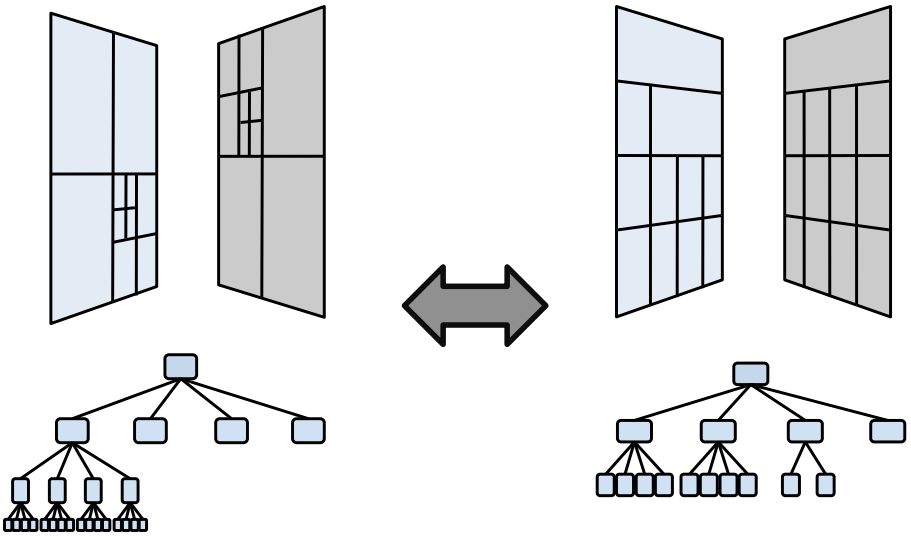


Figura 4.2: A l'esquerra tenim una representació de les divisions que genera el mètode de radiositat jeràrquica i a la dreta tenim una representació de les façanes dels edificis procedurals. Tot i que a simple vista els elements no són molt semblants, observant la seva estructura de dades es pot veure que comparteixen moltes característiques.

node de modelatge procedural pot ser una planta sencera d'un edifici), però la radiositat jeràrquica no depèn d'això, sinó dels factors de forma calculats. Per tant, la manera com es subdivideix un patch en radiositat jeràrquica és completament arbitrària i, per això, es podria fer servir la subdivisió que genera el modelatge procedural. Així doncs, arribem a la conclusió que es podria calcular la radiositat jeràrquica amb les primitives que genera el modelatge procedural. L'avantatge fonamental d'aplicar aquest mètode a la nostra estructura de dades és que, a diferència del mètode pur de radiositat jeràrquica on la subdivisió s'ha de generar, nosaltres aprofitarem la subdivisió pròpia del modelatge procedural ja que existeix des del moment en que s'ha creat l'edifici.

4.2 El procés d'il·luminació

Tot i que la part de radiositat jeràrquica és la més important d'aquest capítol, cal dir que no és l'única. Per poder implementar el mètode és necessari preparar l'estructura de

dades i obtenir la informació necessària pel càlcul. Tal i com es pot veure a la Figura 4.2, hem integrat aquest mètode dins el nostre procés d'il·luminació a fi i efecte d'utilitzar-ne les funcionalitats pel càlcul de l'il·luminació global.

Primer de tot cal saber quines parts dels edificis estan sota la llum directa del sol i identificar-los dins l'estructura de dades. Aquesta cerca es fa mitjançant l'assignació d'identificadors a cadascun dels elements que es renderitzen de tal manera que cada identificador és únic i pertany a un sol node. Situant la càmera a la posició del sol, podem saber quins d'aquests elements estan directament il·luminats. Un cop tenim la llista d'elements visibles des de la font de llum, atorguem a cada node una energia inicial en funció de la seva visibilitat.

Per poder compartir l'energia entre els edificis pròxims cal establir una relació entre ells. Aquests 'links' seran els que ens marcaran quins intercanvis d'energia s'han de fer. Després, s'ha de redistribuir aquesta energia emmagatzemada als nodes entre els elements d'un mateix edifici i, posteriorment, compartir-la amb els edificis adjacents a través dels enllaços existents entre edificis veïns. Finalment, i ja fora del mètode de radiositat jeràrquica, s'ha d'afegir la il·luminació directa per donar un grau més de realisme a l'escena.

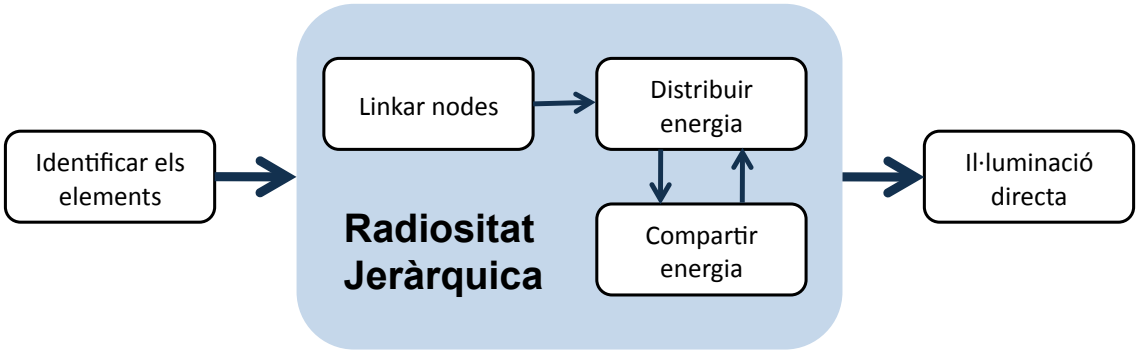


Figura 4.3: Esquema del procés d'il·luminació global. La radiositat jeràrquica n'és la part més important.

4.2.1 Identificació dels nodes

El procés de càlcul de la il·luminació global comença identificant quins elements estan directament il·luminats per la llum del sol. Com hem vist al Capítol 3, els edificis estan representats en un arbre de nodes, on cada node pot tenir fills més petits o pot ser un node fulla, o sigui, ser un node ternimal que no té fills i té la geometria que es finalment es visualitzarà. Aquest últims són els que ens interessen a l'hora de començar el càlcul. Per això cal recorre l'arbre de nodes i assignar un identificador únic a cadascun dels nodes fulla. Aquest identificador ens servirà per reconèixer quins nodes són visibles des del sol i ens permetrà assignar-los una certa energia segons la seva visibilitat. Per saber quins, de tots els nodes fulla, són visibles des de la font de llum, es col·loca una càmera ortogonal a la posició del sol i es fa un render de tota l'escena al FrameBuffer, pintant cada node amb el color referent al seu identificador. Aquesta projecció és ortogonal pel fet que considerem que els rajos del sol arriben paral·lels a l'escena, ja que entenem el sol com una font de llum no puntual i situada a l'infinit. La Figura 4.4 mostra un esquema del mètode que fem servir per identificar els nodes.

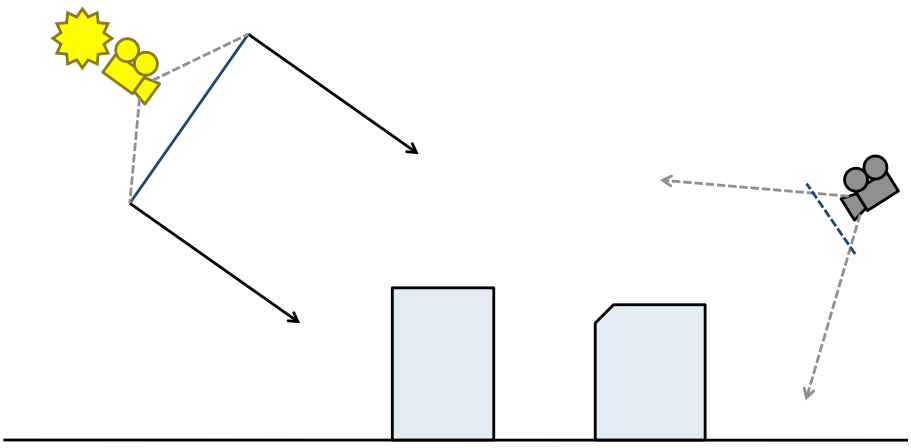


Figura 4.4: El dibuix de la dreta (en gris) representa la posició de la càmera de visualització habitual, amb projecció perspectiva. El dibuix de l'esquerra (en groc) representa la posició de la càmera en el moment de capturar els identificadors, a la mateixa posició que la font de llum.

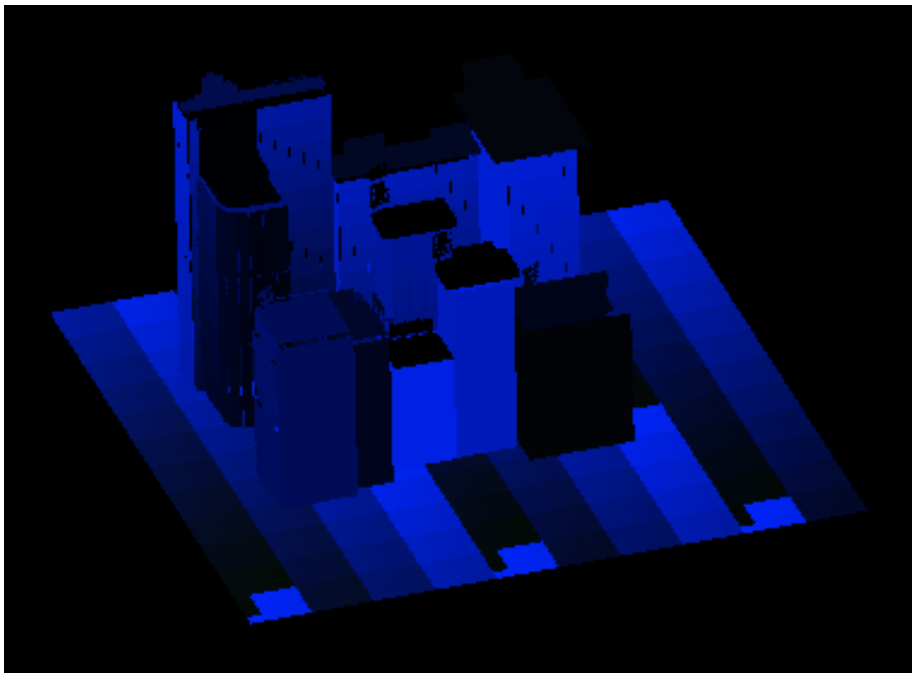


Figura 4.5: Contingut del FrameBuffer durant el render per fer la captura dels identificadors. L'identificador 0 està reservat pel fons perquè els id's comencen per 1.

Per passar els identificadors utilitzem els tres canals de color RGB. Fent servir aquest mètode tenim la possibilitat d'identificar més de 16 milions de nodes diferents. Comptant que cada edifici té una mitjana de 80 nodes visibles, amb aquest mètode podríem renderitzar més de 200,000 edificis sense tenir problemes de memòria.

Per poder obtenir un color per cada node, separem l'identificador en tres parts i assignem cadascuna d'elles a un canal de color. Inicialitzem l'identificador com un *long* de 32 bits així tenim 8 bits per cada canal. Hem decidit fer servir només els tres canals de color tot i que, si fos necessari, podríem utilitzar també el canal *alpha*, que ens donaria molta més capacitat. Els tres colors són normalitzats i enviats a renderitzar a través de la funció d'OpenGL *glColor3f()* (veure Figura 4.6). El resultat d'aquest render es pot veure a la Figura 4.5.

```
if(renderMode == TO_BUFFER){
      int red = id >> 16 & 0xFF;
      int green = id >> 8 & 0xFF;
      int blue = id & 0xFF;
      glColor3f(red/255.0, green/255.0, blue/255.0);
}
```

Figura 4.6: Codi on es mostra la conversió dels identificadors a els tres components de color RGB.

Per obtenir els identificadors dels nodes visibles des del sol, llegim els píxels del FrameBuffer, restaurem l'identificador i fem un simple recompte de quants píxels hi ha de cada color. Per restaurar l'identificador obtenim els tres canals RGB i els ajuntem fent el pas invers a la separació:

$$id = R * 2^{16} + G * 2^{8} + B$$

4.2.2 Assignació de l'energia

Un cop llegit el FrameBuffer i fet el recompte de píxels de cada color, el següent pas consisteix en inicialitzar la radiositat que té cada node visible des de la font de llum. En el càlcul d'aquesta radiositat hi intervenen diferents factors:

$$B_g = \left(\frac{P}{a}\right) * E_f$$

on

- B_g és la radiositat d'entrada del node.
- P és percentatge de visibilitat del node en funció de la mida del viewport
- a és l'àrea de la primitiva del node.
- E_f és l'energia total de la font de llum.

P es calcula en funció de la visibilitat del node des de la font de llum. Hem establert que per cada píxel del FrameBuffer s'assigna 1 fotó al node corresponent. Per simplificar la nomenclatura considerem que 1 fotó representa l'energia associada a un píxel observat des del sol. Així doncs, cada fotó (o píxel) representa el transport d'una unitat d'energia del total emès pel sol. Cada node tindrà tants fotons com píxels del seu color (identificador) siguin vistos des del sol. Tot i això i per tal de desvincular el recompte d'energia de la mida del viewport i de la resolució del FrameBuffer, calculem el percentatge equivalent dels píxels de cada node i fem servir aquest valor en els càlculs posteriors.

4.2.3 Distribució de l'energia

El següent pas consisteix en aplicar el mètode de radiositat jeràrquica vist al Capítol 2. Per fer-ho, cal establir primer les relacions entre els diferents elements dels edificis. Aquestes relacions o links vénen definides per un mètode que determina quins elements dels edificis són viables per a intercanviar energia i quins no. Són diferents factors els que intervenen en aquesta decisió. La Figura 4.7 mostra el pseudocodi de la funció *findLinks*, que s'encarrega d'explorar recursivament l'arbre de dos nodes donats per cercar enllaços entre els seus elements. Tal i com hem vist al Capítol 2 aquesta funció compara els dos nodes i crea un enllaç entre ells en el cas de complir unes determinades característiques. Si no es compleixen, baixa per l'arbre a través dels fills d'un dels nodes, cercant millors candidats per ser enllaçats.

La tasca d'esbrinar si una parella de nodes és vàlida o no, recau en un altre mètode, anomenat *oracle*. La Figura 4.8 mostra el pseudocodi de l'adaptació de la funció *oracle* vista al Capítol 2.

Aquesta funció té en compte bàsicament dos factors a l'hora de decidir si cal crear un enllaç o no. El primer és la distància entre els dos elements candidats: dos nodes que estiguin a una distància molt gran no intercanviaran energia. A l'hora d'implementar el mètode hem decidit posar com a distància mínima una vegada i mitja l'amplada d'un carrer, perquè entenem que la contribució d'un element, més enllà d'aquesta distància, és massa petita com per tenir-la en compte. Així doncs, els edificis compartiran energia

```
function findLinks(Node n1, Node n2){
  Node aux = oracle(n1, n2);
  if(aux != NULL)
    if(aux = n1)
      for( all children from n1 )
        findLinks(child, n2);
    else if(aux = n1)
      for( all children from n2 )
        findLinks(n1, child);
}
```

Figura 4.7: Pseudocodi de la funció findLinks.

```
function oracle(Node n1, Node n2) Return Node{
  float FF12 = formFactor(n1,n2);
  float FF21 = formFactor(n2,n1);
  Node res = NULL;
  if(distance(n1, n2) < MIN_DISTANCE)
    if(FF12 > MIN_FORM_FACTOR)
      res = n1;
    else if(FF21 > MIN_FORM_FACTOR)
      res = n2;
    else
      createLink(n1,n2);
  return res;
}
```

Figura 4.8: Pseudocodi de la funció oracle.

només amb els seus adjacents i amb el terra (veure Figura 4.9).

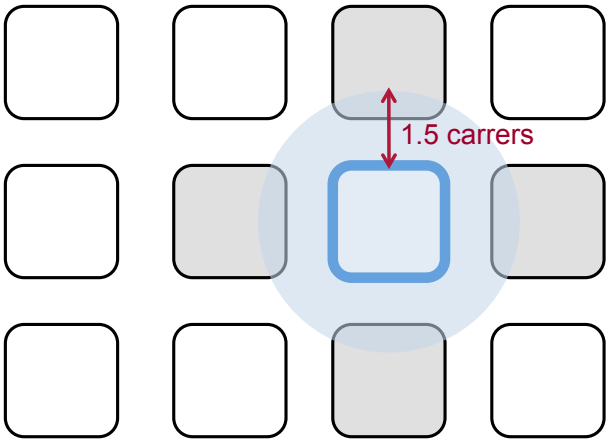


Figura 4.9: Representació en planta d'un model urbà. L'edifici en blau només intercanviarà energia amb els edificis que estiguin dins el seu radi d'acció (1.5 carrers de distància).

L'altre factor que intervé en la decisió de l'*oracle* és el Factor de Forma (Veure Capítol 2). Aquest factor és un paràmetre que quantifica la "qualitat"geomètrica de l'enllaç entre dos elements. Si el Factor de Forma d'un enllaç entre dos nodes és més gran que un determinat llindar, vol dir que s'hauran d'explorar els fills d'un dels nodes per buscar un enllaç de més "qualitat". A la Figura 4.10 podem veure exemples dels enllaços entre els elements de l'escena.

El següent pas consisteix en redistribuir la radiositat que inicialment només tenen els nodes fulla per tot l'arbre, preparant així l'estructura de cada edifici per fer l'intercanvi d'energia amb els seus veïns. D'aquesta tasca se n'encarrega el mètode anomenat *push-pull*. Aquest és un procés indispensable ja que, tot i que al final només es visualitzen els nodes fulla, l'intercanvi d'energia entre edificis es pot produir amb qualsevol dels nodes, a qualsevol nivell de l'arbre. Per això cal redistribuir la radiositat a tots els nodes de l'arbre perquè si no, l'intercanvi només es feria amb els nodes fulla i no consideraríem l'intercanvi d'energia als nivells més alts. Les Figures 4.11 i 4.12 mostren un exemple gràfic del funcionament del mètode.

A la part de *push* (Figura 4.11), cada node rep una aportació del seu pare. La primera

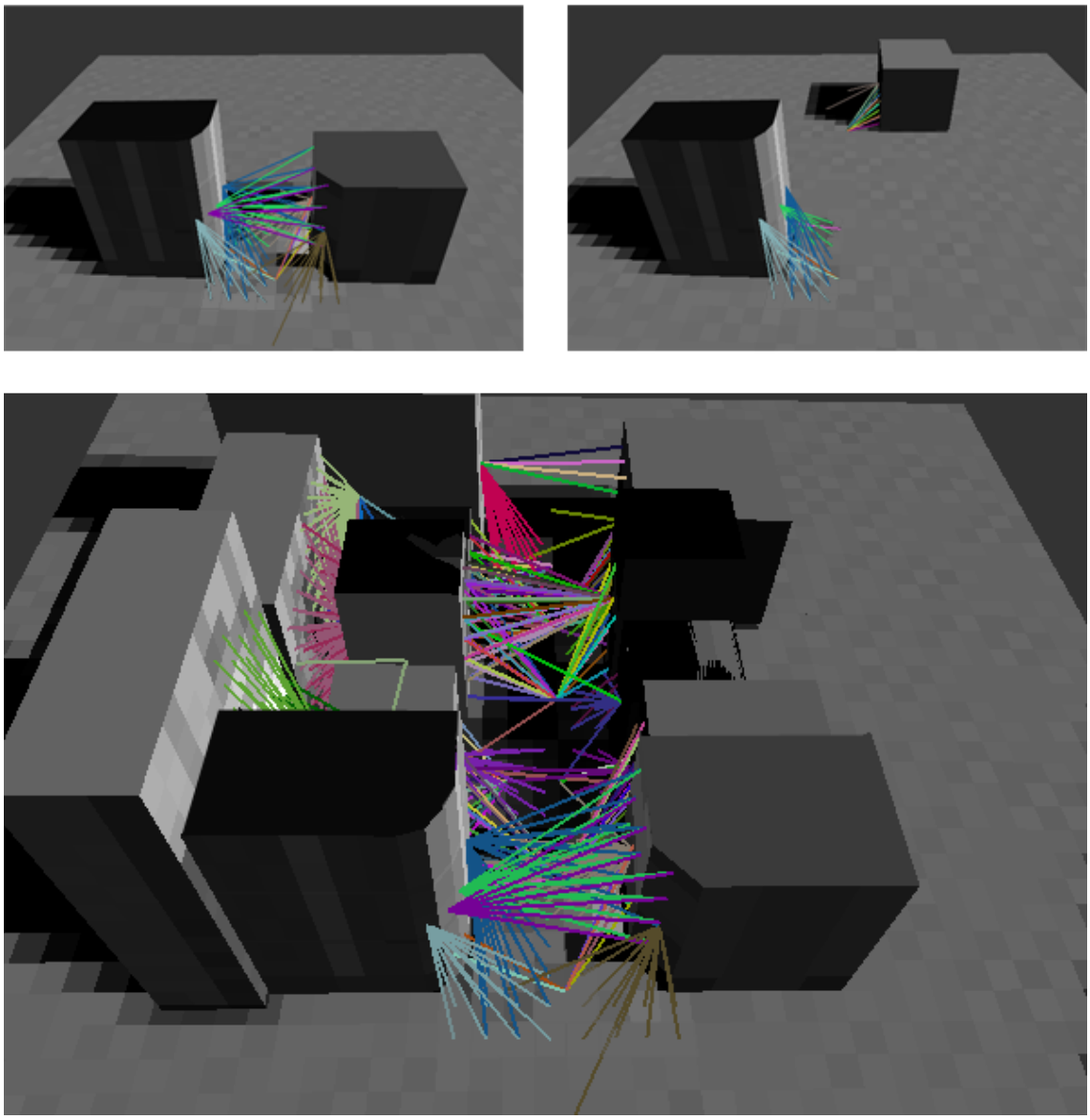


Figura 4.10: La imatge de dalt a l'esquerra mostra els enllaços entre les façanes de dos edificis adjacents i el terra. A dalt a la dreta es pot veure com els edificis que estan més lluny d'una certa distància no generen links entre ells. La imatge de baix mostra un carrer d'una ciutat amb els enllaços entre els edificis.

vegada aquesta aportació és nul·la ja que la radiositat només és als nodes fulla. Tot i ser innecessari la primera vegada, com que l'intercanvi d'energia s'executa vàries vegades (varis rebots de la llum), és important fer aquest pas.

A la part de *pull* (Figura 4.12), la radiositat dels fills es transmet cap als pares, ponderada en funció de la part proporcional de cada fill en relació al seu pare. La Figura 4.13 mostra el pseudocodi del mètode *pushpull*.

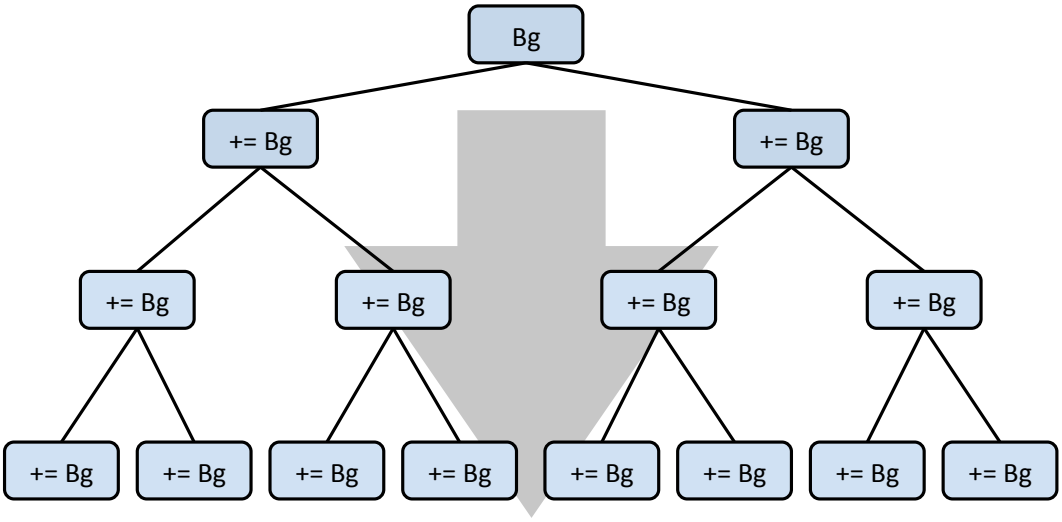


Figura 4.11: La part de *push* del mètode.

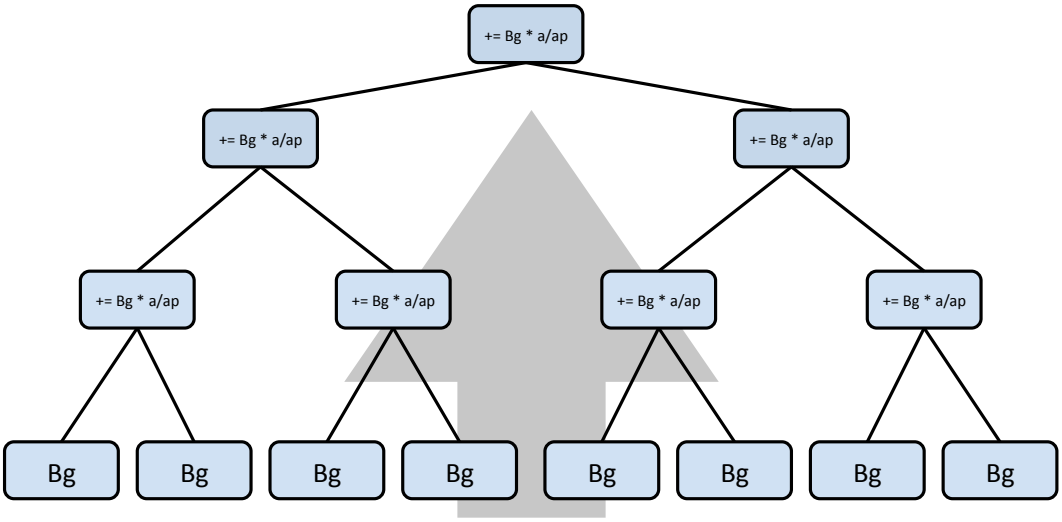


Figura 4.12: La part de *pull* del mètode.

```
function pushPull( Node father, float Bdown ) return float{
  float Bup, Baux;
  if( !father.children )   //node fulla
    Bup = father.Bg + Bdown;
  else
    Bup = 0;
    for( all children )
      Baux = pushPullRecursive( child, child.Bg + Bdown );
      Bup += Baux * ( child.area / father.area );
    father.Bs = Bup;
    return Bup;
}
```

Figura 4.13: Pseudocodi de la funció *pushpull*.

4.2.4 Intercanvi de l'energia

En aquest punt és on comença l'intercanvi d'energia pròpiament dit. Inicialment, els nodes emissors compten amb la radiositat de sortida (Bs) igual a la radiositat d'entrada (Bg). És en el moment de fer l'intercanvi d'energia que s'hi aplica una atenuació causada pel coeficient de refracció del material i pel factor de forma del link que mantenen els dos nodes. La Figura 4.14 mostra la nostra versió del mètode per compartir l'energia entre els elements d'una escena, vist al Capítol 2.

```
function shareRadiosity(){
  for( all links )
    link.n1.Bg += link.FF12 * link.n2.Bs * REFRAC;
    link.n2.Bg += link.FF21 * link.n2.Bs * REFRAC;
}
```

Figura 4.14: Pseudocodi de la funció d'intercanvi d'energia.

Com es pot veure, l'intercanvi d'energia es fa a dues bandes, perquè considerem que

els links són bidireccionals, o sigui que un node és a la vegada emissor i receptor. A la primera passada d'intercanvi només emetran energia aquells nodes que estiguin directament il·luminats, però a les posteriors passades l'energia es repartirà per ambdues bandes. S'ha de procurar resetejar el Bg a cada passada de l'algorisme per no afegir radiositat de més a aquells nodes que ja han compartit la seva energia.

Seguidament a l'intercanvi d'energia cal tornar a redistribuir la radiositat per tots els nodes de l'arbre executant de nou el mètode *pushPull*. Aquests dos passos s'executaran tantes vegades com calgui fins que l'aportació d'energia sigui massa petita per contribuir a la il·luminació d'un node. A la Figura 4.15 es pot veure el resultat del mètode de radiositat jeràrquica.

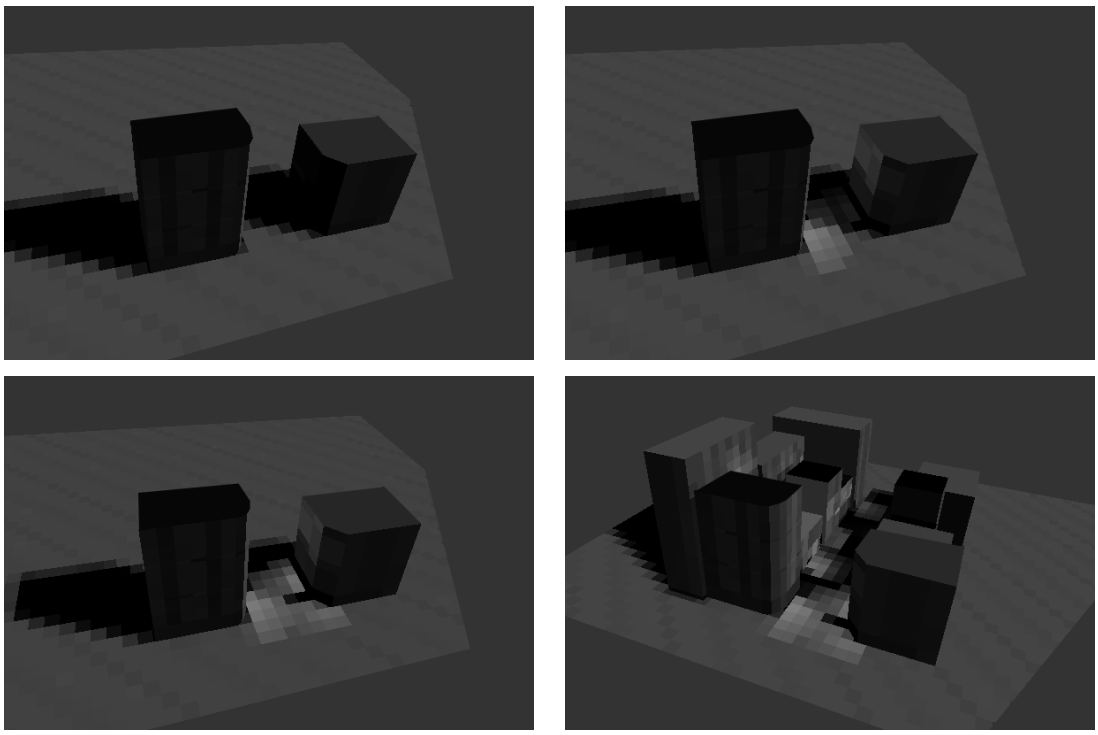


Figura 4.15: A dalt a l'esquerra: l'escena només amb la il·luminació directa. Les zones on no arriba la llum estan totalment a l'ombra. A dalt a la dreta: l'escena amb un pas d'intercanvi d'energia. A baix a l'esquerra: l'escena amb dos passos de l'algorisme. A baix a la dreta: l'escena final amb dos passos de l'algorisme.

4.2.5 Il·luminació directa: Shadow Mapping

En aquest punt, el càlcul de la il·luminació indirecta ja està acabat, i cada node té l'energia que li pertoca. Per obtenir un resultat més realista però, fa falta afegir la il·luminació directa provinent de la font de llum. Aquesta il·luminació directa genera, per definició, unes zones totalment il·luminades i altres zones en ombra. A diferència del càlcul indirecte, que genera ombres de baixa freqüència, ara necessitem molta més resolució per poder tenir ombres perfectament definides i realistes (d'alta freqüència) perquè en el càlcul indirecte l'element més petit té la mida d'una primitiva i és massa gran per utilitzar-ho com a ombra real.

Per aquesta raó, la millor opció és utilitzar Shadow Mapping. La idea bàsica d'aquest mètode és la de comparar distàncies. Mirant l'escena des de la font de llum podem saber quins elements són visibles pel sol o, al revés, quins estan amagats per altra geometria i per tant, estan a l'ombra. La tècnica de Shadow Mapping consisteix en comprovar, per cada punt de l'escena, si esta directament il·luminat o no. Això es fa comprovant si algun altra element de l'escena tapa la visó d'aquest punt. Per fer aquestes comprovacions s'utilitza un mapa de profunditats, que és bàsicament una imatge en escala de grisos (un sol canal de color) capturada des de la font de llum on cada píxel té el valor de la distància entre un punt de la geometria i el sol (Veure Figura 4.16). Aquesta imatge l'hem generat en temps de pre-procés, durant l'etapa de d'obtenció dels identificadors dels nodes. Aprofitant la lectura del FrameBuffer també hem obtingut el DepthMap i l'hem guardat per poder-hi accedir en temps d'execució a l'hora de generar el Shadow Mapping.

Al moment de renderitzar l'escena, per cada fragment a pintar, es comprova si la distància a la font de llum és superior al valor guardat de la seva projecció al depthMap. En cas afirmatiu, voldrà dir que aquest punt està a l'ombra d'algun altre element de l'escena i per tant, s'atenuarà el seu color final.

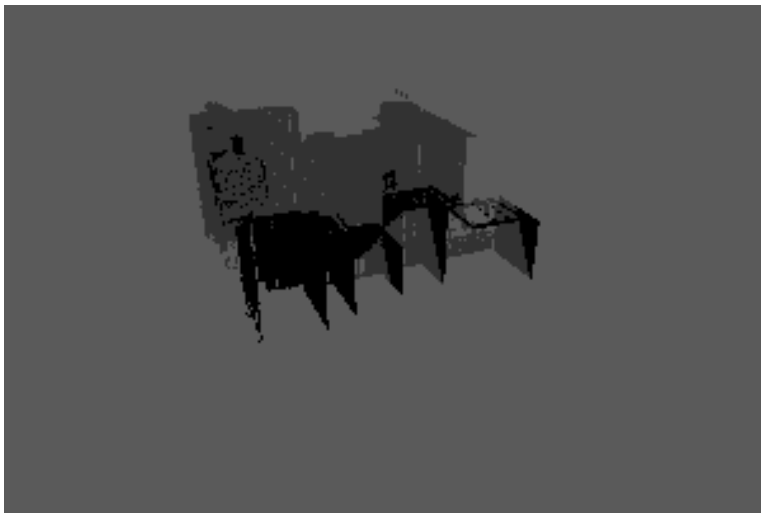


Figura 4.16: DepthMap d'una escena. El pla de retallat posterior tindrà valors propers al 1.0 i el pla de retallat anterior tindrà valors propers al 0.0. El render al DepthMap es fa només amb les cares interiors dels polígons per evitar problemes precisió a l'hora de consultar el valor de punts molt propers al límit de l'ombra.

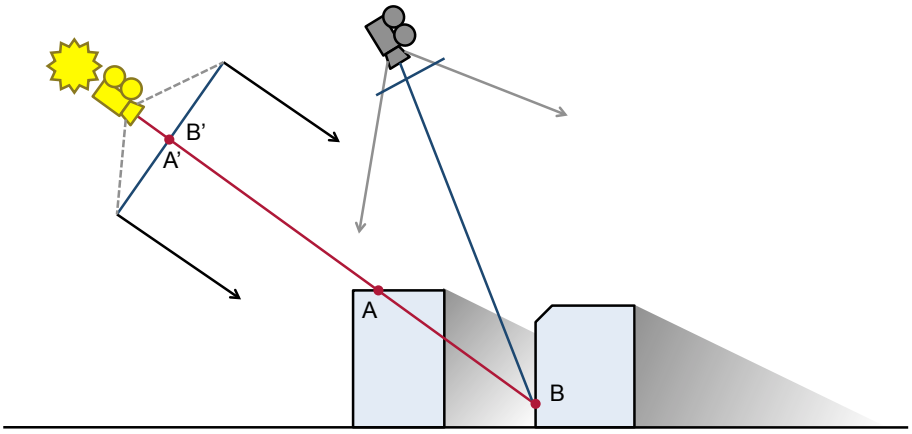


Figura 4.17: A i B es projecten sobre el mateix píxel però com que A està més a prop del sol, serà aquest punt el registrat al deptMap. En el moment de pintar el punt B a l'escena, comprovarem si la component de profunditat de B és igual al valor de la seva projecció al depthMap. Si el valor del mapa de profunditats és més petit, llavors voldrà dir que B està a l'ombra d'algun altre element.

4.2.6 Render final

Per obtenir la imatge final és necessari combinar la il·luminació indirecta i la directa de tal manera que es pugui aconseguir l'escena final perfectament il·luminada. Per aquest últim pas hem decidit utilitzar *shaders*. Fer la combinació de llum utilitzant la GPU ens permet treballar a nivell de fragment i millorar l'aplicació del Shadow Mapping.

L'únic color que enviem a pintar a través de les comandes *glColor()* d'OpenGL és la radiositat de cada node, obtinguda mitjançant el mètode de radiositat jeràrquica. Aquest valor arriba a la GPU, juntament amb el depthMap de l'escena. És aquí on es comprova si un determinat fragment està situat a l'ombra o és directament il·luminat per la font de llum.

Per la programació de la GPU hem desenvolupat els nostres propis shaders que ajuden a millorar el càlcul de la il·luminació de l'escena.

El *Vertex Shader* rep un vèrtex i el fa servir per obtenir informació útil que servirà per al càlcul de les ombres. Per una part, s'obté la normal del vèrtex, multiplicant la NormalMatrix pel vèrtex. El producte escalar entre la normal i la direcció de la llum servirà per modificar el color en funció de la posició de la llum.

L'altre pas important és transformar el vèrtex en relació a la font de llum. Aquest valor *ShadowCoord* s'interpolarà per ser enviat al *Fragment Shader*. A la Figura 4.18 es pot veure el pseudocodi del *Vertex Shader*.

```
Vertex Shader{
  normal = normalize(NormalMatrix * Vertex);
  lightDir = normalize(lightPos);
  NdotL = max(dot(normal, lightDir), 0.0);
  ShadowCoord = lightMatrix * Vertex;
  FrontColor = NdotL * Color;
}
```

Figura 4.18: Pseudocodi del *Vertex Shaders*.

El *Fragment Shader* s'encarrega de saber si un determinat fragment és a l'ombra o no. Per això, consulta el ShadowMap amb les coordenades interpolades del fragment que ha rebut del *Vertex Shader*. Si la distància del fragment a la font de llum és més petita que la component de profunditat del fragment, el color final s'atenuarà a la meitat. El color final del fragment és la combinació del color de cada primitiva, obtingut a partir del mètode d'il·luminació global, i de l'atenuació de l'ombra, en funció de l'energia total de la font de llum i de l'àrea del fragment.

```
Fragment Shader{
  DistanceFromLight = texture2D(shadowMap, shadowCoord);
  if( distanceFromLight < shadowCoord.z )
    shadow = 0.5;
  else
    shadow = 1.0;

  FragColor = FrontColor + ( shadow * Ef * Afrag );
}
```

Figura 4.19: Pseudocodi del *Fragment Shader*. *Ef* és l'energia total del a font de llum i *Afrag* és l'àrea del fragment.

5. Resultats i Discusió

En aquest capítol podrem veure el resultat final del nostre projecte. Mostrarem com funciona l'algorisme que hem creat aplicat a diferents escenes i amb diferents posicions de la font de llum. També podrem veure l'eficiència de l'algorsime en diferents condicions així com altres dades significatives de l'execució del mètode.

5.1 Render

El funcionament del nostre mètode és indepentent de la posició del sol i, per tant, aquest no influeix en el procés d'intercanvi d'energia. No importa la quantitat de nodes que estiguin il·luminats inicialment ja que l'intercanvi es fa unilateralment per tots els nodes que estiguin enllaçats. Les Figures 5.1, 5.2, 5.3 i 5.4 mostren la mateixa escena, capturada des de la mateixa posició, però amb diferents posicions de la font de llum. Es pot veure com la il·luminació canvia, tan la directa, generada amb el Shadow Mapping, com la indirecta, obtinguda a partir de la nostra implementació del mètode de radiositat jeràrquica. Cal dir que si l'aplicació permetés canviar la posició del sol en temps d'execusió, no caldria recalcular els links entre elements, perquè aquests depenen només de la geometria de l'escena, no de la posició de la font de llum.

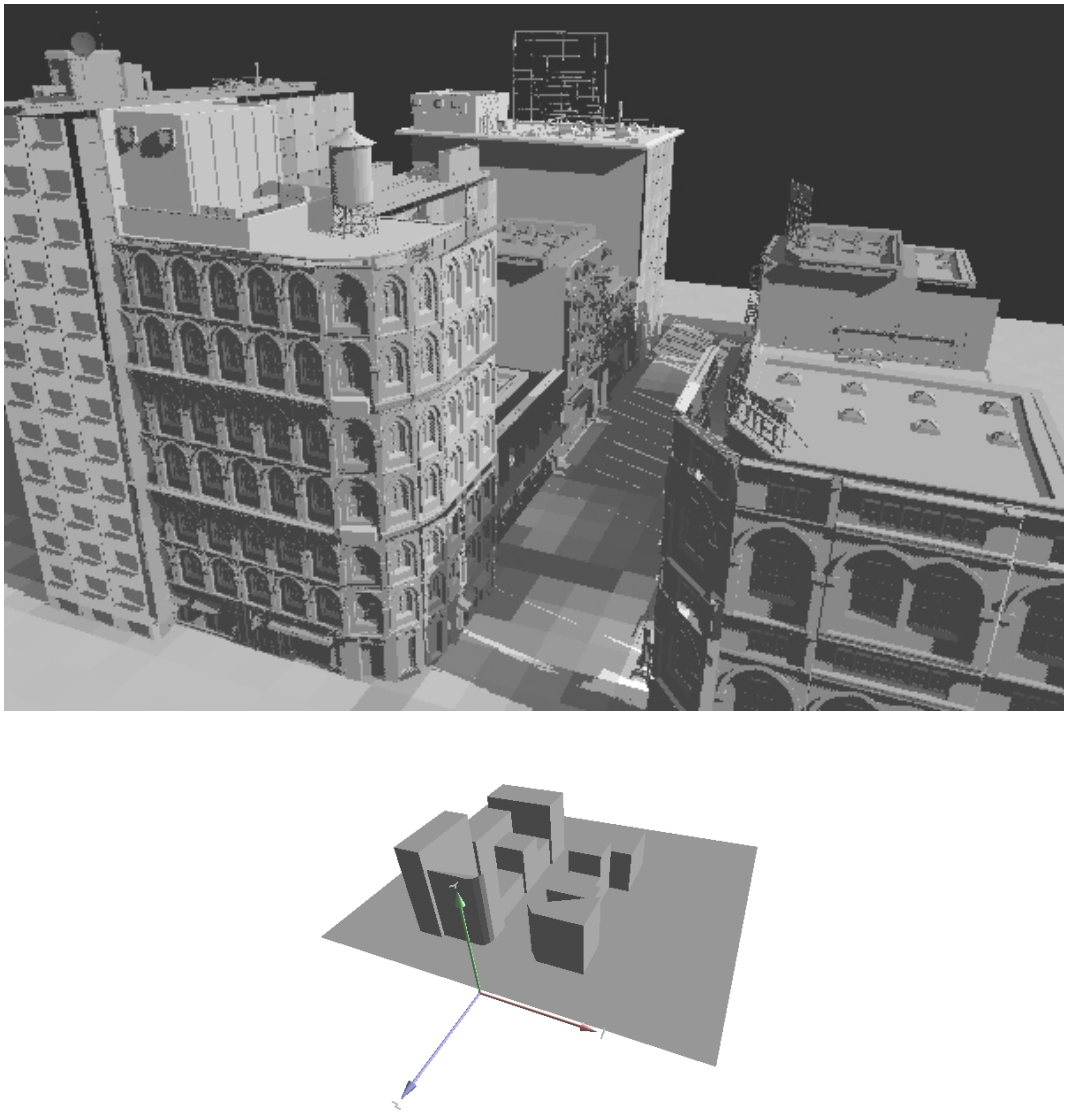


Figura 5.1: Escena general amb la direcció del sol a (35, 30, 0).

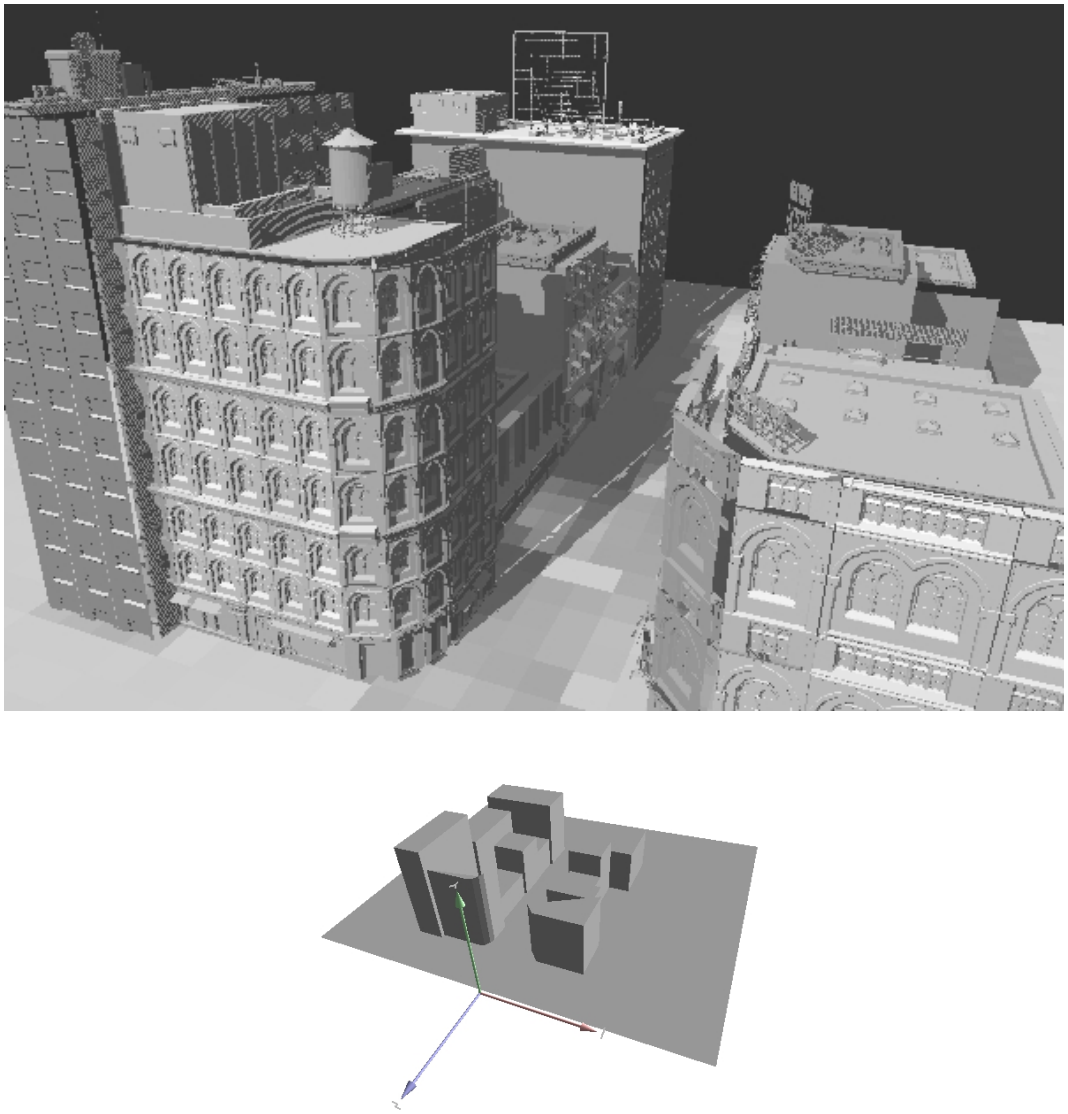


Figura 5.2: Escena general amb la direcció del sol a (-10, 30, 30).

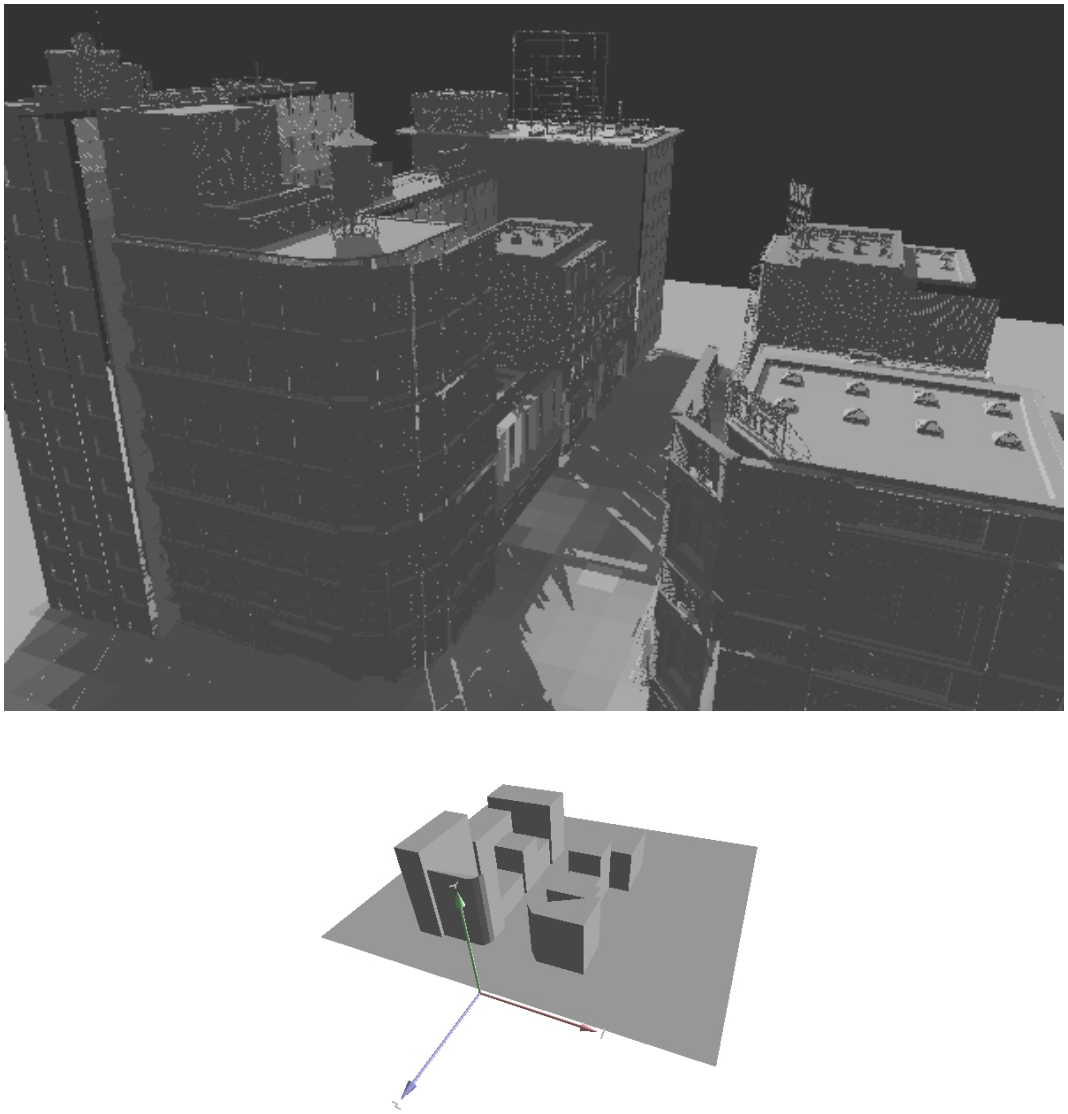


Figura 5.3: Escena general amb la direcció del sol a (-20, 30, -30).

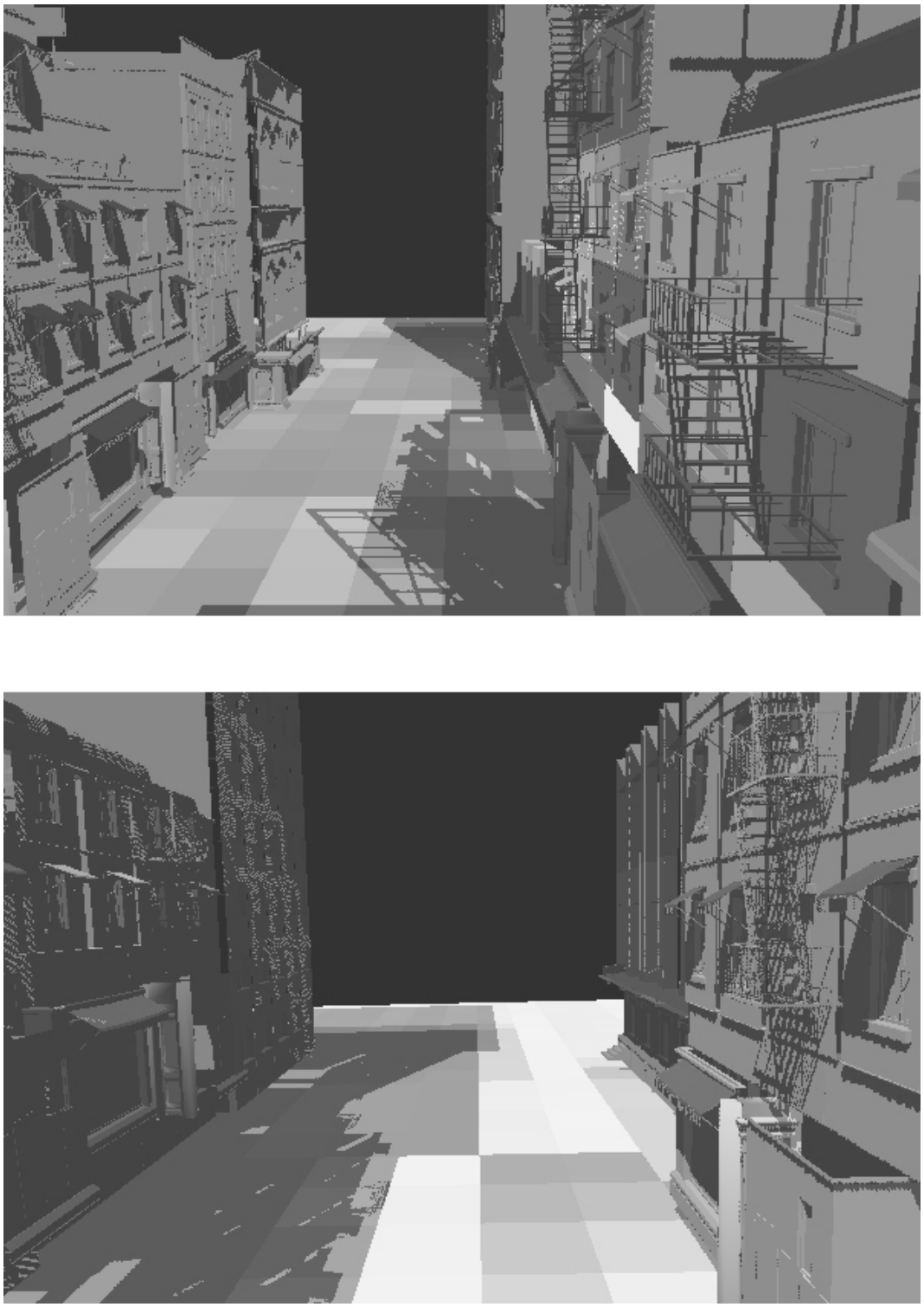


Figura 5.4: Dues captures a peu de carrer. Cal notar que a les zones on es projecta l'ombra directa del Shadow Mapping, les primitives del terra són d'un color més fosc que les que estan il·luminades pel sol. Tot i que aquestes primitives reben energia de les seves veïnes, sempre estaran més fosques que estan les directament il·luminades.

5.2 Dades

El càlcul de la il·luminació global es fa durant el pre-procés i, tot i que el temps no és tan important com en l'execució, cal obtenir un rendiment acceptable per tal de no haver d'esperar molt a l'hora d'utilitzar el sistema. La següent taula recull els temps de pre-procés en relació a la quantitat de primitives de l'escena. Com es pot veure gràficament a la Figura 5.5, l'increment del temps és aproximadament lineal en relació el número d'edificis.

Pel que fa el rendiment en temps d'execució, amb l'escena més gran (77,440 polígons) obtenim una mitjana de 10 FPS durant la visualització.

Número d'edificis	Número de primitives (aprox.)	segons
1	80	0,209
2	160	0,281
3	240	0,339
4	320	0,392
5	400	0,453
6	480	0,481
7	560	0,507
8	640	0,536
9	720	0,562
10	800	0,604
11	880	0,649
12	960	0,695

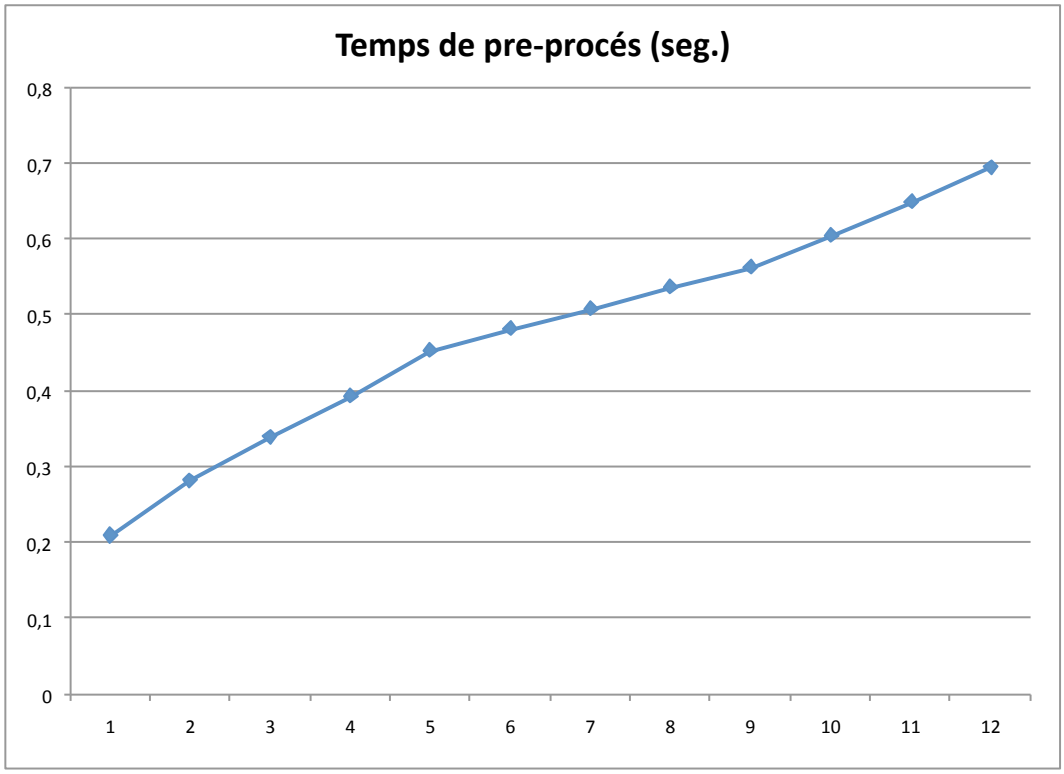


Figura 5.5: Gràfic que mostra els temps de pre-procés en segons, en relació al número d'edificis

Els temps de pre-procés que hem vist a la taula anterior depen en gran part de la quantitat d'enllaços entre primitives que hi hagi a l'escena. Com hem vist al Capítol 4, la creació d'aquests enllaços depèn sobretot de la distància entre els elements candidats. Aquesta distància no és arbitrària. El formigó, que és el material més habitual en un model urbà, té un índex de reflexió d'entre 0.2 i 0.3. Això fa que la quantitat d'energia que pot compartir un edifici no sigui gaire gran. Si a aquest fet hi afegim que l'energia es dissipa en funció de la distància, tenim que l'aportació que pot fer un element a un altre, situat a una certa distància, no té massa influència sobre el resultat final.

Cal dir però que, en el cas que afegíssim superfícies especulars a l'escena, com ara miralls o vidres, hauríem de reconsiderar aquesta distància perquè l'index de reflexió d'aquests materials és molt més alt i per tant, l'aportació a la il·luminació dels elements més llunyans seria prou gran per tenir-la en compte.

Un altre factor que influeix en el rendiment del sistema és el càlcul dels rebots de la llum. Per tal d'obtenir més realisme, el mètode de radiositat jeràrquica s'executa repetides vegades, simulant els diferents rebots dels rajos de llum entre els elements. Tot i que a la realitat, la llum es reflexa moltes vegades, per problemes de costos computacionals, nosaltres hem de limitar aquests càlculs. De nou, cal establir un llindar a partir del qual considerem que l'aportació que pot fer un element a un altre no és prou significatiu com per tenir-lo en compte. Per decidir el llindar òptim hem calculat l'error relatiu que hi ha entre les radiositats d'entrada (B_g) de tots els nodes, en els successius passos d'execució:

$$E_r^i = \sqrt{\sum_{nodes} (B_g^i - B_g^{i-1})^2}$$

La següent taula i la Figura 5.6, recullen aquest error:

Passos d'execució	Error relatiu
1	8,108135058
2	3,862198949
3	3,70112842
4	3,69030425

Com es pot veure, a partir del tercer pas d'execució la diferència baixa molt. Per tant doncs, assumim que a partir de la segona passada de l'algorisme, l'intercanvi d'energia entre els elements d'una escena ja no és significatiu.

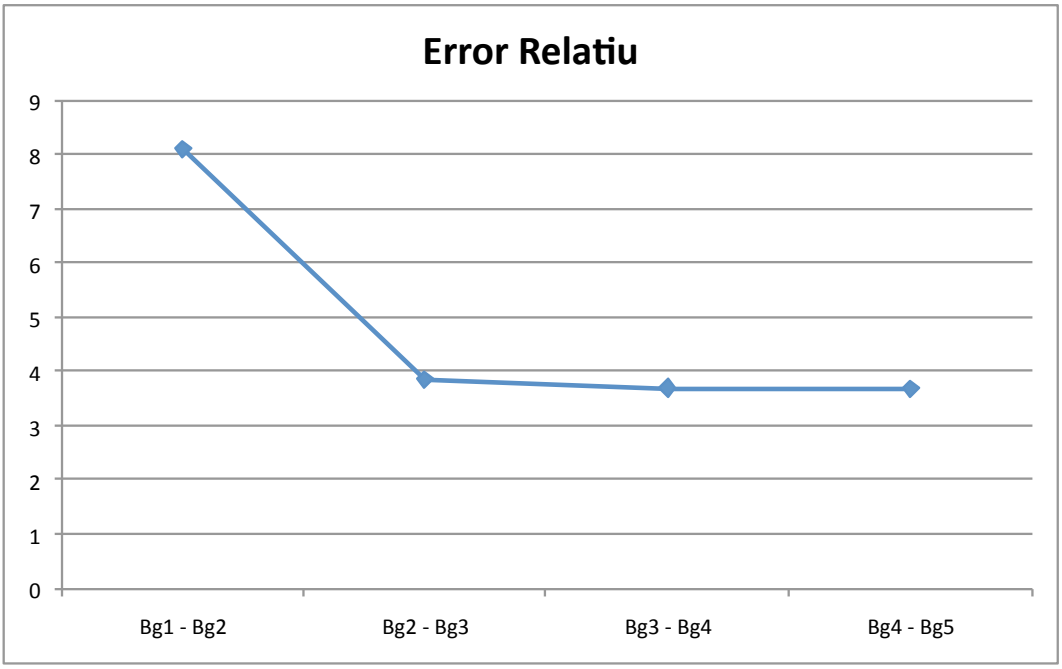


Figura 5.6: Gràfic de l'error relatiu entre els diferents passos d'execució de l'algorisme de radiositat jeràrquica, sobre el valor de la radiositat d'entrada d'un node.

6. Conclusions i treball futur

Per concloure aquest projecte podem dir que hem aconseguit crear un sistema robust per fer el càlcul de la il·luminació global per a qualsevol model urbà donat. Tot i fer les proves amb el model d'un carrer, podem assegurar que el sistema pot suportar models més grans, amb les restriccions implícites de cada màquina. Hem aconseguit un algorisme que aprofita les característiques dels models procedurals i n'obté un resultat satisfactori.

De cares a millorar el sistema hi ha alguns punts on seria recomanable aprofundir més:

- **Millorar el sistema de càrrega de models**: La idea de carregar els models a través de fitxers XML és bona i funciona correctament, però alguns edificis queden desajustats per problemes de precisió. Caldria revisar el sistema de càrrega dels models geomètrics (assets) perquè sigui capaç de corregir aquests errors.

- **Interpolar la radiositat**: Amb el sistema actual, la radiositat es calcula a nivell de primitiva, és a dir, tota la primitiva s'acaba pintant d'un únic color pla. A la realitat, aquest tipus d'ombres no existeixen i per tan caldria millorar aquest aspecte. Per això proposem crear una solució on el color de cada primitiva sigui el resultat de la interpolació amb el color de les seves veïnes, a fi i efecte d'obtenir una ombra homogènia i sense salts bruscs de color.

- **Il·luminació dinàmica**: Permetre canviar la posició del sol en temps d'execució i, a més, tenir en compte tot el cel com a possible font de llum difusa, no només el sol.

- **Textures**: Aplicar textures a la geometria per tal de fer més creïble l'escena.

- **Ambient Occlusion**: Tot i que la il·luminació que hem calculat dóna bons resultats, seria apropiat fer un pas més i aplicar alguna altra tècnica d'il·luminació realista com ara *Ambient Occlusion* que donarien molta més credibilitat a les escenes. Aquesta tècnica podria aplicar-se, fins i tot, en espai de textura calculant-ho fora del sistema i guardant el resultat a les textures de la geometria de cadascun dels assets.

Bibliografia

[1] Michael F. Cohen, John Wallace, and Pat Hanrahan. *Radiosity and realistic image synthesis*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[2] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 197–206, New York, NY, USA, 1991. ACM.

[3] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006.

[4] Y. Parish and P. Müller. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, 2001.

[5] G. Patow. Interactive modelling of procedural buildings. In *XX CONGRESO ESPAÑOL DE INFORMÁTICA GRÁFICA, CEIG2010*, pages 197–206, 2010.

[6] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[7] R. Ridorsa and G. Patow. The **skyline**Engine system. In *XX CONGRESO ESPAÑOL DE INFORMÁTICA GRÁFICA, CEIG2010*, pages 207–216, 2010.

[8] J. Sun, X. Yu, G. Baciu, and M. Green. Template-based generation of road networks for virtual city modeling. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 33–40, New York, NY, USA, 2002. ACM.

[9] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Transactions on Graphics*, 22(4):669–677, july 2003. Proceeding.