
PROJECTE FINAL DE CARRERA

# Reconstrucció 3D de superfícies sobre GPU a partir de múltiples vistes

Estudis: Enginyeria de Telecomunicació

Autor: Marc Maceira Duch

Directors: Jordi Salvador i Josep Ramon Casas

Barcelona, Juny 2011

# Resum del Projecte

El present projecte presenta una implementació sobre targetes gràfiques (GPU) d'un algorisme de reconstrucció de superfícies 3D a partir de múltiples vistes. Es busca una estratègia de paral·lelització eficient que permeti visualitzar la reconstrucció en temps real. S'ha triat un entorn de desenvolupament genèric (OpenCL) per arquitectures multiprocessador, de manera que el codi desenvolupat no depengui del hardware de la targeta gràfica específica.

L'abast del projecte comprèn el tractament de les imatges 2D utilitzades en la reconstrucció, la reconstrucció de superfície, un post-processament de la superfície i la integració de les diverses parts en una aplicació en temps real.

# Resumen del Proyecto

El presente proyecto presenta una implementación sobre tarjetas gráficas (GPU) de un algoritmo de reconstrucción de superficies 3D a partir de múltiples vistas. Se busca una estrategia de paralelización eficiente que permita visualizar la reconstrucción en tiempo real. Se ha escogido un entorno de desarrollo genérico (OpenCL) para arquitecturas multiprocesador, para evitar que el código desarrollado dependa del hardware de la tarjeta gráfica específica.

El proyecto contempla el tratamiento de las imágenes 2D utilizadas en la reconstrucción, la reconstrucción de superficie, un post-procesamiento de la superficie y la integración de las diferentes partes en una aplicación en tiempo real.

# Abstract

This project presents an implementation for graphics processing unit (GPU) of an algorithm to reconstruct 3D surfaces from multiple views. It seeks an efficient parallelization strategy to visualize the reconstruction in real time. We have chosen a development environment (OpenCL) for generic multiprocessor architectures to prevent that the code development depends on the specific GPU hardware.

The scope of the project includes the treatment of 2D images used for reconstruction, the 3D surface reconstruction, surface post-processing and the integration of the different parts in a real time application.

# Índex

# Índex de figures

# Capítol 1

# Introducció

Una possibilitat als sistemes amb múltiples vistes consisteix en la reconstrucció tridimensional (3D) dels elements de l'escena. Aquests elements es calculen a partir de les dades de les múltiples vistes, podent assolir millors resultats en augmentar el nombre de càmeres. Amb la reconstrucció 3D d'una escena aconseguim una representació alternativa a les diferents vistes. La reconstrucció permet concentrar la informació de les diferents càmeres en una única representació, a partir de la qual es pot visualitzar l'escena des de qualsevol punt de vista, així com servir com a suport per realitzar processat de més alt nivell, com pot ser la detecció, reconeixement i seguiment dels elements de l'escena o l'anàlisi dels esdeveniments que es puguin produir.

## 1.1 Motivació

El procés de generar la reconstrucció 3D d'una escena es considera una primera etapa bàsica en l'anàlisi multivista. La informació de les múltiples càmeres es fusiona obtenint una única representació de l'escena. Aquesta representació serveix com a base per realitzar processos de més alt nivell. Per aquest fet, és necessari disposar d'un mètode de reconstrucció de superfícies a temps real.

En els darrers anys s'ha produït un ràpid desenvolupament de les capacitats de càlcul de les targetes gràfiques (*Graphics Processing Units* o *GPU*s) degut a les demandes computacionalment intensives dels videojocs. Aquesta evolució ha permès disposar d'una gran capacitat de càlcul a un preu assequible. Els fabricants, conscients de les possibilitats creixents de la computació paral·lela,

han fet accessible la programació de les targetes gràfiques per a aplicacions de propòsit general. Aquest desenvolupament de les targetes gràfiques les fa ideals per desenvolupar-hi aplicacions en temps real, on es necessita processar i presentar resultats sense retards. Algorismes amb gran quantitat de càlculs repetitius es podran paral·lelitzar i executar eficientment en *GPU*, en canvi algorismes amb dependències de càlculs anteriors no es podran portar eficientment a les targetes gràfiques. Per tant, en el desenvolupament d'aquest projecte haurem de dedicar especial atenció a la utilització d'algorismes que puguin explotar correctament aquestes característiques.

## 1.2 Objectius

El Grup de Processament d'Imatge i Vídeo treballa en la recerca de nous mètodes de reconstrucció 3D. Es busca aconseguir una representació alternativa del primer pla d'una escena que permeti recuperar la informació de les diferents vistes sense pèrdues rellevants, o el què és el mateix, es pot tornar a les vistes originals per projecció de la reconstrucció 3D. La reconstrucció ha de permetre comprimir la informació amb un cost computacional raonable.

La tesi d'en Jordi Salvador 'Surface Reconstruction for Multi-View Video'[7] proposa un mètode de reconstrucció que compleix aquests requisits, basat en un mostreig aleatori de l'espai en busca de punts de superfície.

L'objectiu d'aquest projecte final de carrera és aconseguir una reconstrucció 3D de superfícies en temps real d'una escena real capturada per les múltiples càmeres. Es considerarà que l'aplicació treballa a temps real si és capaç de processar un nombre de reconstruccions proper a deu per segon. Amb aquest nombre d'actualitzacions la taxa de refresc serà suficient per tenir una percepció de continuïtat de moviment. Per tal d'aconseguir aquest objectiu es proposa una implementació sobre targeta gràfica.

Per tal d'aconseguir realitzar la reconstrucció de superfícies necessitarem implementar un seguit de funcionalitats que es resumeixen a continuació i que es desenvoluparan en capítols posteriors:

- Projecció: relacionar punts 3D de l'escena amb els punts 2D de cada càmera
- Desdistorsió: les lents de les càmeres introdueixen distorsió que caldrà corregir

- Segmentació de primer pla: la reconstrucció 3D requereix d'una segmentació a nivell d'imatge

  - Post-processament: per tal de millorar la segmentació s'utilitzen tècniques de morfologia matemàtica

- Reconstrucció 3D de superfície: trobar punts de superfície i una orientació per cada punt. Amb el mètode utilitzat requereix:

  - Generació de nombres aleatoris: el mètode mostreja l'espai aleatòriament

  - Càlcul de vectors propis: l'orientació dels punts es calcula a partir dels valors i vectors propis

- Acolorir els punts: es desitja que cada punt tingui un color representatiu de la superfcie real que s'està mostrejant, de manera que la representació sigui útil tant per visualització com per un tractament posterior de les dades

- Representació de l'escena: renderitzar la representació basada en punts en un entorn que ens permeti la seva visualització i manipulació interactiva

## 1.3 Estructura

El present document s'organitza per temàtiques. El capítol 2 presenta la programació en entorns multiprocessador i la plataforma escollida. A continuació, els tres següents capítols resumeixen els conceptes teòrics en que es basa el present projecte final de carrera, i presenten els detalls més rellevants en la seva implementació en un context de *GPUs*.

Un cop presentats els conceptes i la implementació realitzada, es detallen els resultats assolits i la seva anàlisi. Seguidament es mostra l'entorn i el *software* utilitzat en la implementació de la aplicació en temps real.

Finalment es conclou amb les conclusions i possibles millores del treball realitzat.

# Capítol 2

# GPGPU: General-Purpose computing on Graphics Processing Units

La primera decisió a prendre en el context del projecte és la d'escollir la plataforma de programació utilitzada. Es tracta d'una decisió estratègica del Grup de Processament d'Imatge i Vídeo, ja que en el futur es pretén seguir potenciant l'ús de la capacitat de computació de les targetes gràfiques. L'experiència al començar aquest treball al Grup d'Imatge es limitava al desenvolupament d'una aplicació basada en CUDA [8]. La decisió passava per seguir amb CUDA o buscar alguna alternativa. Es va considerar OpenCL com l'alternativa més viable a CUDA. Tot seguit es presenta una breu revisió de la programació en targetes gràfiques que justifica els motius per optar per OpenCL.

## 2.1 Evolució de la programació en *GPU*

Les exigències actuals dels videojocs i els entorns de disseny gràfic ha portat als fabricants de targetes gràfiques a augmentar dràsticament la seva capacitat de càlcul. El processament gràfic en general no requereix d'un processament seqüencial, de forma que els fabricants han optat per paral·lelitzar el seu hardware, dissenyant dispositius amb un nombre cada cop més elevat de processadors. Actualment les *central processing unit* (*CPU*) també s'estan dissenyant amb un nombre creixent de processadors, ja que s'està arribant al límit de velocitat dels processadors. Per tant, les estratègies de paral·lelització cada cop són més habi-

tuals.

El desenvolupament d'OpenGL, una *application programming interface* (API) que permet accedir a la targeta gràfica per generar imatges de gran qualitat, permetia als programadors accedir a un nombre creixent de possibilitats i exigia un esforç als fabricants de targetes gràfiques, sent necessaris canvis en l'estructura del hardware per aconseguir millors rendiments.

El creixement de la capacitat de computació per part de les targetes gràfiques va fer pensar en poder utilitzar aquests dispositius per aplicacions de propòsit general (*General-Purpose computing on Graphics Processing Units - GPGPU*). La figura 2.1 mostra l'evolució de la capacitat de càlcul entre *CPU* i *GPU*.

NVIDIA, conscient que *GPGPU* podia implicar un increment de vendes de *GPU*s i sent conscient dels problemes d'accés a les *GPU*s va començar a desenvolupar un *Software development kit* (SDK) per tal de simplificar la programació *GPGPU*. El resultat d'aquest desenvolupament és CUDA (*Compute Unified Device Architecture*), presentat al novembre del 2006.

CUDA permet la cooperació de la *CPU* i els processadors de la *GPU* en un sol programa, creant un nou paradigma de computació anomenat computació heterogènia. Els desenvolupadors de software poden programar aplicacions de propòsit general a través de "C for CUDA", que es bàsicament C amb algunes extensions de NVIDIA i algunes funcionalitats de C++. CUDA ha estat àmpliament usat en diferents àrees com simulacions científiques, mèdiques, processament del senyal, criptografia o processat d'imatge.

A remolc de l'èxit de CUDA, ATI també va desenvolupar un SDK per tal d'accedir a les seves targetes gràfiques anomenat Stream SDK, però no ha tingut el mateix èxit que CUDA.

L'any 2008 Apple Inc, comença a treballar amb un nou sistema amb la idea de proposar-lo com a estàndard per a la programació paral·lela i l'anomena Open Computing Language (OpenCL). Per tal de desenvolupar-lo com a estàndard el proposa al Khronos Group[11]. Aquest grup és un consorci entre vàries empreses enfocat a desenvolupar estàndards oberts i lliures de *royalties* finançat pels seus membres. Aquest grup s'ocupa del desenvolupament d'estàndards com són OpenGL, OpenVG, OpenKODE, OpenML entre altres. Formen part del grup empreses com AMD, Apple, Intel, NVIDIA, Ericsson, Nokia, Oracle, Sony. Les primeres especificacions d'OpenCL van ser presentades a finals del 2008 i les primeres implementacions daten de l'any 2009.

OpenCL permet la programació paral·lela sobre plataformes heterogènies con-

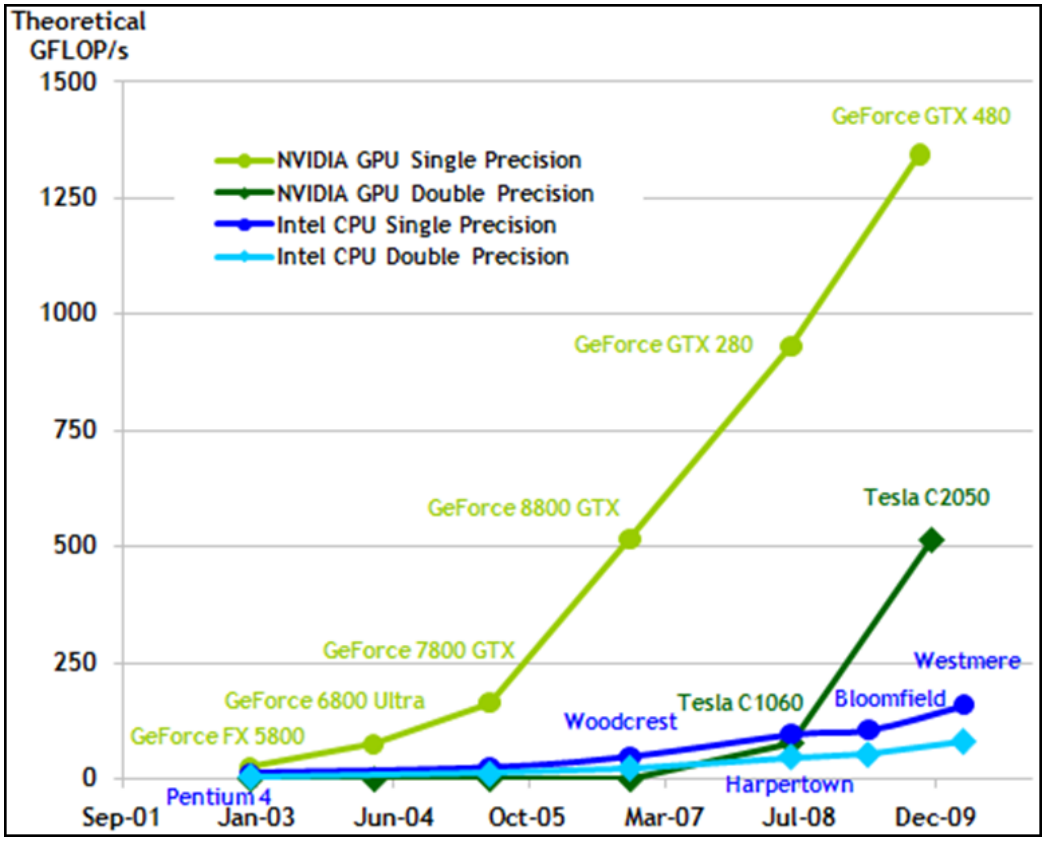


Figura 2.1: Comparació FLOPS (*FLoating point Operations Per Second*) entre *CPU* i *GPU*

sistents en *CPU*s, *GPU*s i altres processadors, ampliant les possibilitats en no centrar-se en la programació en targetes gràfiques com passa en CUDA. OpenCL inclou el llenguatge de programació, basat en C99 per l'escriptura de les funcions o *kernels* a executar, i API's per tenir el control de les diverses plataformes.

Un cop presentada l'evolució de la programació en targetes gràfiques dels últims anys, estem en disposició d'exposar els factors que ens van fer decidir per OpenCL en lloc de CUDA.

## 2.2 OpenCL vs CUDA

En primer lloc cal destacar que l'objectiu era realitzar una implementació eficient no depenent del hardware. L'evolució constant de les targetes gràfiques implica canvis en l'estructura del hardware i una capacitat de càlcul cada cop més gran.

L'experiència amb CUDA al Grup de Processament d'Imatge i Vídeo indicava la possibilitat d'optimitzar el codi per una targeta gràfica en concret, amb diferents opcions de memòria i de la forma en què s'hi accedeix. Aquests aspectes poden comportar que l'optimització per una targeta gràfica en particular no sigui la mateixa que per targetes de nova generació, així com esforços per millorar la implementació d'un algorisme més que millorar l'algorisme en sí.

L'altre aspecte clau a considerar és l'estat actual de la tecnologia i l'evolució futura. Com s'ha comentat anteriorment, la tecnologia dominant en l'actualitat és CUDA. Aquest fet es deu als esforços fets per NVIDIA en el desenvolupament de CUDA. No només va ser llançada abans que OpenCL sinó que, en ser una tecnologia propietària, el suport per part de NVIDIA és molt bo, tant pel que fa a solucions de possibles problemes com al llançament d'eines que permeten la detecció d'errors i l'optimització del codi. En ser optimitzada per la estructura del hardware d'NVIDIA, el rendiment és superior a OpenCL en general. D'altre banda, actualment té una comunitat de desenvolupadors més gran, la qual cosa ajuda a trobar solucions més ràpidament.

Per contraposició, OpenCL es troba en un estat de desenvolupament menys madur, però en tractar-se d'un estàndard fa pensar en una major implantació futura. El concepte de programació heterogènia està pensat per tal de poder ser executat en diversitat de plataformes formades per una combinació de *CPU*s, *GPU*s i altres processadors, i està pensada en l'evolució dels processadors en dispositius multinucli, independentment si es tracta de *CPU*s, *GPU*s o altres.

Es va optar per OpenCL per el major potencial futur d'aquesta tecnologia davant del millor rendiment actual de CUDA, la possibilitat de poder usar dispositius de diferents fabricants i el rendiment no dependent de la arquitectura de la targeta gràfica en particular.

## 2.3 Característiques d'OpenCL

### 2.3.1 Model de plataforma

El model de plataforma d'OpenCL consisteix en un *host* connectat a un o més dispositius OpenCL (OpenCL *devices*). El *host* s'encarrega de la creació de la infraestructura necessària per l'execució de l'aplicació així com el control de l'execució. Els *devices* executen el codi indicat pel *host*. Habitualment en la nostra execució disposarem d'una *CPU* que actuarà com a *host*, mentre que la *GPU*

actuarà com a *device*. Aquest seria el model més senzill, però es pot disposar de vàries *GPU*s connectades a un *host* o que el mateix *host* actuï a la vegada com a *host* i com a *device*.

Els *devices* OpenCL estan dividits en una o més *compute units* (CUs) que a la vegada es divideixen en un o més *processing elements* (PEs). Cada *compute unit* coordina l'execució d'una part del codi OpenCL en *processing elements*. La figura 2.2 mostra el model de plataforma d'OpenCL.

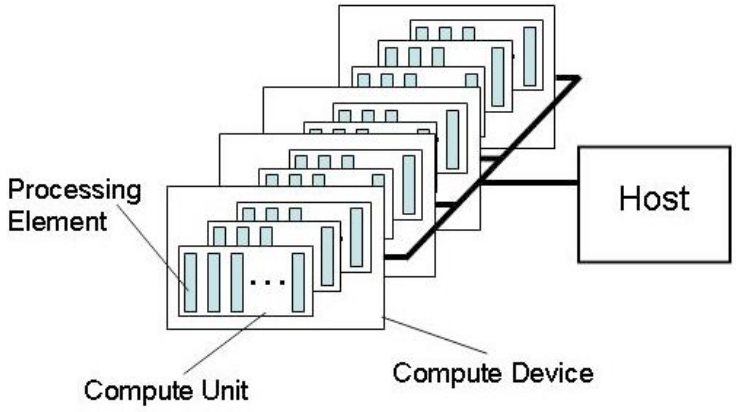


Figura 2.2: Model de plataforma OpenCL. Un *host* connectat a diversos *devices*.

### 2.3.2 Model d'execució

L'execució dels programes amb OpenCL es divideix en dues parts: els *kernels* s'executen als *devices* OpenCL i el programa *host* s'executa al *host*. El programa *host* defineix el context d'execució pels *kernels* i dirigeix la seva execució. Per tal de mostrar el model d'execució d'una aplicació amb OpenCL, partirem d'un exemple senzill com és la binarització d'una imatge. Els passos a seguir en pseudocodi pel programa *host* són els següents:

- Llegir la imatge a binaritzar
- Crear el context d'execució OpenCL: preparar els *devices*, compilar el codi OpenCL, definir les dimensions dels *kernels* a executar
- Crear les estructures de memòria necessàries: buffer on llegir/escriure la imatge
- Escriure la imatge a binaritzar al buffer d'entrada

- Enviar el *kernel* de binarització a executar
- Llegir la imatge resultant

El nucli del model d'execució d'OpenCL està definit per com s'executen els *kernels*. Quan el *host* envia un *kernel* per la seva execució es defineix un espai d'índexs. Cada *kernel* s'executa un cop per cada punt d'aquest espai d'índex. Cada execució s'anomena *work-item* i està identificada pel seu punt a l'espai d'índex. Cada *work-item* executa el mateix codi però l'execució està marcada per l'índex.

Seguint amb l'exemple anterior el codi OpenCL de la binarització d'una imatge seria el següent:

```
__kernel void binarize(   __global uchar * image,
                          uchar threshold,
                          uint2 dims)
{
  uint2 coords=(uint2)(get_global_id(0),get_global_id(1));
  uint i = buffer_access_2d(dims,coords);

  if( image[i] >= threshold )
      image[i]=255;
  else
      image[i]=0;
}
```

Aquest codi serà executat un cop per cada píxel de la imatge. S'organitzaran els *work-items* en dues dimensions, corresponents al nombre de píxels en cada dimensió de la imatge. Al programa *host*, es fixen el nombre de píxels en cada dimensió. La primera crida del codi serveix per identificar el *work-item* dins de l'espai d'índex. Aquest identificador, en aquest cas, ens servirà per accedir a la posició corresponent del buffer, però tamb el podríem usar per diferenciar el comportament del codi a les diferents zones de la imatge a tractar.

Els *work-item* es troben organitzats en *work-groups*. Cada *work-group* s'executa en una mateixa *compute unit* i els *work-items* d'aquest s'executen concurrentment en els diversos processing units que tingui la *compute unit*. Cal tenir en compte l'organització en *work-groups* per diferents motius:

- Les dimensions dels *work-groups* indicades afectaran al rendiment de l'aplicació. Usualment *work-groups* majors assoleixen millor rendiments.

- Si el nombre d'elements en alguna dimensió no és divisible per la mida del *work-group*, alguns *work-items* no hauran d'executar codi ja que accediríem a posicions de memòria incorrectes. El programador ha de controlar aquest comportament.
- La sincronització de diferents *work-items* només és possible dins d'un mateix *work-group*.
- Un *work-group* no allibera les *compute units* utilitzades fins que tots els *work-items* han acabat. Cal balancejar el treball entre els *work-item* per tal que el més lent del *work-group* no condicioni el temps d'execució final.

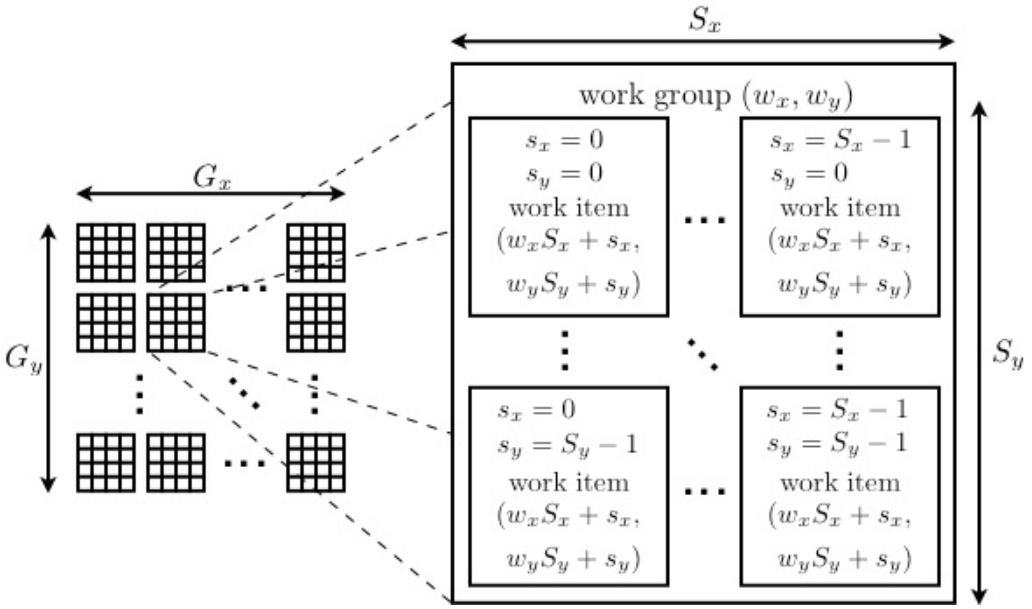


Figura 2.3: Execució de *kernels*. Divisió entre *work-groups* i *work-items*

La figura 2.3 mostra un *kernel* de dos dimensions amb dimensions globals $\mathbf{G_x}$, $\mathbf{G_y}$ i dimensions locals $\mathbf{S_x}$, $\mathbf{S_y}$. Cada *work-item* s'identifica mitjançant índexs dins del *global-group* i del *local-group*.

### 2.3.3 Model de memòria

Els *work-items* executant-se en un *kernel* poden accedir a quatre tipus diferents de memòria:

- Memòria global: permet la lectura i l'escriptura per tots els *work-items* a totes les posicions de memòria de l'objecte.

- Memòria constant: el *host* reserva i inicialitza aquest tipus de memòria i no pot ser modificat pels *work-items*.
- Memòria local: Memòria local dins d'un *work-group*. És compartida per tots els *work-items* d'un *work-group*.
- Memòria privada: memòria reservada per un *work-item*. Les variables definides dins d'un *work-item* formen part de la memòria privada i no poden ser consultades per altres *work-items*.

Per concloure el capítol, recordar que s'ha elegit OpenCL davant CUDA pensant en les possibilitats futures d'una i altra i per la disponibilitat d'usar hardware de diferents fabricants. La programació en OpenCL ofereix majors capacitats de càlcul però incorpora noves característiques que cal tenir en compte al desenvolupar codi. Per tal d'introduir la problemàtica de la paral·lelització, s'ha caracteritzat breument OpenCL, destacant la separació entre *host* i *devices* així com un model d'execució que distribueix el programa entre el *host* i els diferents *devices*.

# Capítol 3

# Projecció i model de càmera

El primer pas per treballar amb múltiples càmeres és poder relacionar la informació de les diferents vistes[2], i per això cal disposar d'un model de càmera. Aquest model és necessari tant per determinar el procés de projecció (paràmetres intrínsecs) com la relació de la càmera amb l'entorn (paràmetres extrínsecs).

## 3.1 Model de càmera pinhole. Projecció i desdistorsió

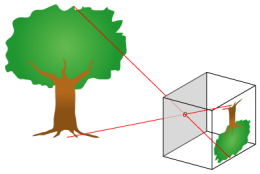

Figura 3.1: Model de càmera *pinhole*

Una pràctica habitual en visió per computador és caracteritzar la càmera amb un model *pinhole*. Aquest model descriu la relació matemàtica entre un punt 3D i la projecció al pla de la imatge quan la càmera es modela com una caixa negra simple, sense lent, amb una petita obertura per on la llum passa i projecta la imatge invertida a la cara oposada de la caixa. La figura 3.1 mostra el model de càmera *pinhole*.

La projecció es realitza sobre un pla seguint un únic punt de projecció. Els paràmetres intrínsecs queden determinats per la distància del centre de projecció

al pla de projecció (distància focal f) i l'adaptació del punt principal del pla (p) al sistema de coordenades de la imatge. La figura 3.2 mostra la relació entre els dos sistemes de coordenades i la projecció d'un punt de l'espai al pla de la imatge.

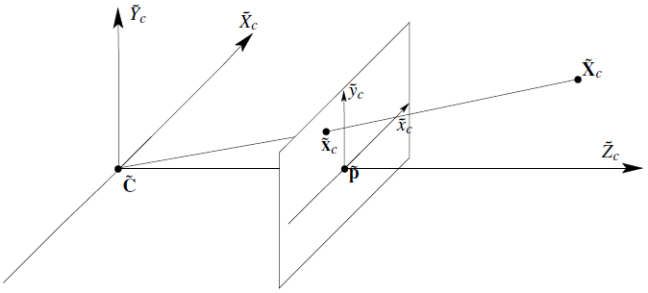


Figura 3.2: Sistemes de coordenades en un model de càmera *pinhole*

D'aquesta forma és possible passar de coordenades de l'espai $\tilde{\mathbf{X}}_c$ a coordenades de la imatge $\tilde{\mathbf{x}}_c$ a través d'una matriu de calibració $\mathbf{K}$:

$$\tilde{\mathbf{x}}_\mathbf{c} = \begin{pmatrix}\tilde{x}_c\tilde{y}_c\tilde{z}_c\end{pmatrix} = \begin{pmatrix}f_x & 0 & p_x 0 & f_y & p_y 0 & 0 & 1\end{pmatrix}\begin{pmatrix}\tilde{X}_c\tilde{Y}_c\tilde{Z}_c\end{pmatrix} = \mathbf{K}\tilde{\mathbf{X}}_\mathbf{c}$$

Amb la matriu de calibració, queda determinada la càmera intrínsicament. Per tal de relacionar-la amb l'espai serà necessari relacionar les coordenades de l'espai amb les coordenades de la càmera. En general, per aconseguir aquesta relació són necessàries una translació i una rotació tal com mostra la figura 3.3.

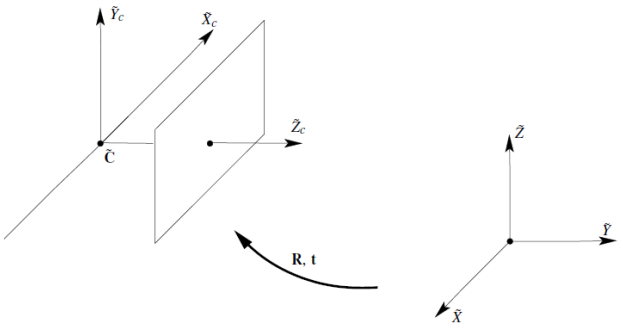


Figura 3.3: Relació de coordenades de la càmera amb les coordenades de l'espai

Matemàticament:

$$\tilde{\mathbf{X}}_\mathbf{c} = \mathbf{R}\tilde{\mathbf{X}} + \mathbf{t}$$

Substituint a l'expressió anterior:

$$\tilde{\mathbf{x}}_\mathbf{c} = \mathbf{K}\tilde{\mathbf{X}}_\mathbf{c} = \mathbf{K}(\mathbf{R}\tilde{\mathbf{X}} + \mathbf{t})$$

Operant es poden agrupar els termes en una matriu de projecció **P**:

$$\tilde{\mathbf{x}}_\mathbf{c} = \mathbf{K}(\mathbf{R}\tilde{\mathbf{X}} + \mathbf{t}) = \mathbf{K}[\mathbf{R}|\mathbf{t}]\tilde{X} = \mathbf{P}\tilde{X}$$

Un cop calibrada la matriu **P** es poden convertir les coordenades de l'espai **X** en coordenades de la imatge $\tilde{\mathbf{x}}_c$. Si les coordenades homogènies en la imatge són (u,v,1)=$\mathbf{x}_i$, es compleix que $z\mathbf{x}_i = \tilde{\mathbf{x}}_c$. És a dir, cal dividir les coordenades $\tilde{\mathbf{x}}_\mathbf{c}$ per la profunditat z per trobar les coordenades del píxel.

Figura 3.4: Imatge original i imatge desdistorsionada

Amb el model *pinhole* es poden caracteritzar la projecció de les escenes al pla de la imatge així com la relació amb les coordenades de la imatge. El model però no té en compte la distorsió de lent. Aquesta distorsió s'ha de compensar per tal de recuperar les coordenades de la imatge correctament i evitar errors en les coordenades dels objectes de la sala. La distorsió més freqüent en les lents de les càmera és la distorsió radial. Tot i això, el model de distorsió usat habitualment és més genèric, compensant també la distorsió asimètrica. El model de distorsió considerat és el següent:

$$\begin{cases} \check{x} = p_x + L(r)(x - p_x) + P_1(2(x - p_x)^2 + r^2) + 2P_2(x - p_x)(y - p_y) \\ \check{y} = p_y + L(r)(y - p_y) + P_1(2(y - p_y)^2 + r^2) + 2P_2(x - p_x)(y - p_y) \end{cases}$$

on: (ǎ,ǐ) són els punts de la imatge distorsionats (x,y) són els punts de la imatge ideals (sense distorsió) ($p_x$,$p_y$) és el centre de distorsió de la imatge r és el radi de distorsió al punt (distància de ($p_x$,$p_y$) a (x,y) ) L(r) = 1 + $k_2r_2$+$k_4r_4$ és la funció de distorsió lineal $P_1$, $P_2$ són els coeficients de distorsió de descentrat

A la figura 3.4 es pot observar l'efecte de la distorsió de lent en la curvatura de les parets de la imatge i la seva correcció un cop compensada la distorsió.

Una forma eficient i habitual de treballar és aplicar el model invers de distorsió a les imatges capturades, de manera que a partir d'aquest moment es pot suposar que no hi ha distorsió. Aplicant la distorsió sobre les imatges s'aconsegueix que

al projectar punts 3D sobre les imatges no s'hagi de compensar la distorsió cada cop.

Per un major aprofundiment en el model de càmera i la forma d'extreure els paràmetres es pot consultar el projecte d'Òscar Garcia [1].

## 3.2 Implementació

### 3.2.1 Transferència de paràmetres *CPU-GPU*

Com s'ha presentat anteriorment, el model de càmera queda determinat per uns paràmetres intrínsecs i extrínsecs que s'agrupen en matrius. Aquests paràmetres es calibren un cop s'han instal·lat les càmeres i s'emmagatzemen en un arxiu.

Per tal de disposar del model de càmera a la *GPU* és necessari disposar d'una forma còmoda de transferir els paràmetres que permeti un bon rendiment. Per tal de facilitar aquestes transferències s'han agrupat els paràmetres necessaris en una estructura. A l'annex A es defineixen alguns detalls rellevants del marc de treball desenvolupat per a la programació amb OpenCL en el context d'aquest projecte. L'estructura CameraCL mencionada en aquest apartat es troba definida a l'annex.

- La matriu de projecció *proj* s'organitza amb una sola dimensió on s'ordenen les tres files i les quatre columnes
- La matriu *cal* emmagatzema els paràmetres intrínsecs de la càmera: el nombre de píxels per unitat de distància en cada dimensió i la posició pel punt principal de la imatge
- La matriu de distorsió *distortion* conté els paràmetres de distorsió $k_2$, $k_4$, $P_1$ i $P_2$
- El nombre de files i de columnes de les imatges captades per la càmera s'indiquen en im_width i im_height

Amb l'estructura definida podrem accedir als diferents camps de la càmera còmodament. S'ha desenvolupat una funció per tal d'inicialitzar els diversos camps d'una CameraCL a *CPU*.

L'estructura a OpenCL es pot definir en memòria global o en memòria privada. En la global tots els *work-items* comparteixen una posició de memòria mentre que en la local cada *work-item* disposa d'una estructura CameraCL privada. En el cas d'utilitzar memòria global es pot definir constant, de forma que pugui ser llegida però no modificada. Diverses proves utilitzant les diferents memòries permeten determinar que s'obtenen millors prestacions (major velocitat d'execució, degut a la reducció d'accessos a memòria global, que és més lenta) amb la memòria privada. D'aquesta forma al treballar amb estructures de dades s'utilitzarà sempre memòria privada.

### 3.2.2 Projecció

L'operació de projecció per una càmera pinhole hem vist que es pot resumir en:

$$\tilde{\mathbf{x}}_\mathbf{c} = \mathbf{K}(\mathbf{R}\tilde{\mathbf{X}} + \mathbf{t}) = \mathbf{K}[\mathbf{R}|\mathbf{t}]\tilde{X} = \mathbf{P}\tilde{X}$$

La matriu de projecció per una càmera en concret es trobarà en el camp proj de la CameraCL. Per tal de trobar les coordenades de la imatge $\tilde{\mathbf{x}}_\mathbf{c}$ a partir d'un punt de l'espai $\tilde{\mathbf{X}}$ cal realitzar la multiplicació per la matriu $\mathbf{P}$. Al tractar-se d'una operació molt utilitzada s'ha creat una funció que a partir d'un punt 3D in i la càmera on es vol projectar el punt, retorna la coordenada 2D de la imatge.

El punt 3D es representa amb un vector de quatre *floats* ja que amb OpenCL 1.0 no existien els vectors de 3 elements. La versió 1.1 d'OpenCL permet vectors de 3 elements però internament segueix utilitzant vectors de 4 elements, motiu pel qual no s'ha canviat la implementació a vectors cl_float3. El vector de 4 coordenades permet realitzar la multiplicació $\mathbf{P}\tilde{X}_4$ directament. Amb un vector de 3 coordenades no seria possible, caldria realitzar una multiplicació seguida d'una suma: $\mathbf{R}\tilde{X}_3 + t$ .

### 3.2.3 Desdistorsió

La distorsió d'un punt queda determinada per l'equació:

$$\begin{cases} \breve{x} = p_x + L(r)(x - p_x) + P_1(2(x - p_x)^2 + r^2) + 2P_2(x - p_x)(y - p_y) \\ \breve{y} = p_y + L(r)(y - p_y) + P_1(2(y - p_y)^2 + r^2) + 2P_2(x - p_x)(y - p_y) \end{cases}$$

El nostre objectiu és compensar aquesta distorsió, per la qual cosa aplicarem la transformació inversa per recuperar la imatge sense distorsió.

La matriu *distortion* de CameraCL conté els paràmetres $k_2$, $k_4$, $P_1$ i $P_2$ i la matriu cal conté els paràmetres intrínsecs $f_x$, $f_y$ $p_x$ i $p_y$. La desdistorsió s'ha encapsulat en una funció que a partir d'un píxel d'entrada *in* retorna un píxel de sortida desdistorsionat. La funció rep com a paràmetre d'entrada una coordenada 2D codificada amb *floats*, tot i que s'acostuma a treballar amb coordenades enteres.

Una primera utilitat que s'aconsegueix amb el model de distorsió implementat és desdistorsionar una imatge. Es busca per a cada píxel de la imatge ideal (sense distorsió), la posició que correspon a la imatge distorsionada aplicant el model de

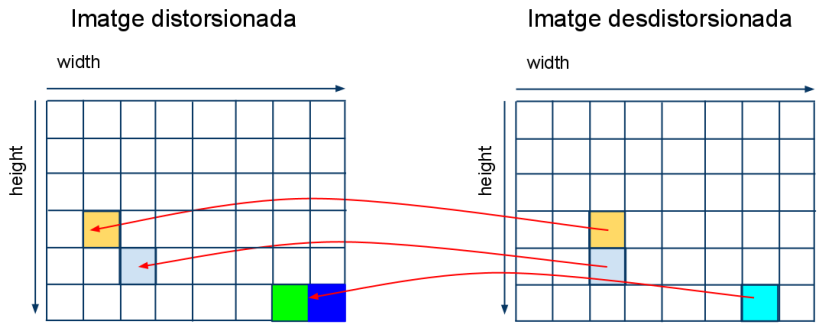


Figura 3.5: Representació del procés de desdistorsió d'una imatge. A l' última fila s'observa una interpolació lineal entre dos píxels. Cada píxel desdistorsionat es resultat de la interpolació dels 4 píxels més propers de la imatge original

distorsió. S'assigna al píxel sense distorsió el resultat d'interpolar bilinearment els 4 píxels més propers.

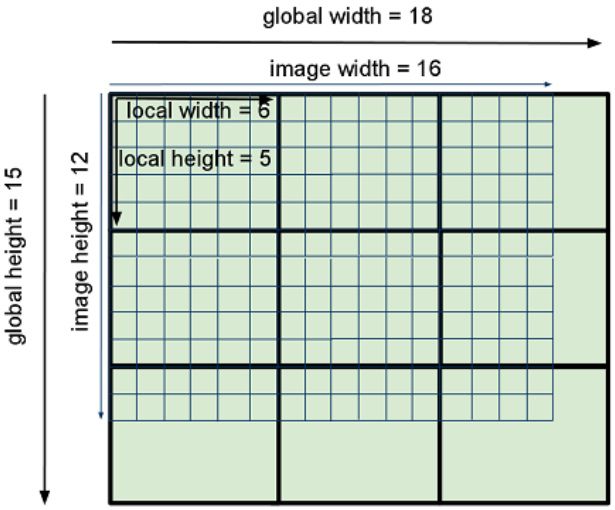


Figura 3.6: Distribució del treball entre els diversos *work-items*. S'observen zones on els *work-items* cauen fora de la imatge

Al processar cada píxel de la imatge destí assegurem que cada píxel es correspongui a un i únic color. Aquest fet no es compliria si distorsionéssim cada punt de la imatge origen, ja que, possiblement, diferents píxels origen correspondrien a un sol píxel destí. Altres píxels no tindrien cap color assignat.

OpenCL permet crear estructures de dades específiques per a imatges proporcionant un millor rendiment i diverses utilitats. Una d'elles és poder seleccionar el *sampler* a utilitzar. El *sampler* determina com s'accedeix a una imatge en un *kernel*, especifica el tipus d'adreçament, com es gestionen les coordenades fora del

rang de la imatge, el mode de filtrat i si les coordenades tenen valors normalitzats o no.

Particularment és interessant el mode de filtrat, permetent elegir entre la coordenada més propera o realitzar una interpolació entre les mostres més properes. A la figura 3.5 es pot observar el procés de desdistorsionar una imatge.

La realització del *kernel* OpenCL implementat assigna cada píxel a un *work-item*. La figura 3.6 mostra una imatge de 12 files i 16 columnes distribuïda en *work-groups* de 5 files i 6 columnes on cada *work-item* processa un píxel. La mida global ha de ser múltiple de la mida local, per tant alguns *work-items* no han de realitzar cap acció ja que accedirien a posicions de memòria no controlades. L'annex A conté la forma elegida de controlar l'execució dels diferents *kernels*.

En aquest capítol s'ha definit el model de càmera amb el qual es treballa en els següents capítols, la possibilitat de corregir la distorsió de lent en les imatges i la relació d'un punt a l'espai 3D amb les coordenades d'imatge, que utilitzarem en la reconstrucció.

# Capítol 4

# Segmentació de primer pla

La segmentació de primer pla és un mètode àmpliament usat per detectar objectes en moviment a partir de càmeres estàtiques. L'aproximació al problema es basa en detectar canvis en l'escena a partir de la diferència entre la imatge actual i una imatge de referència que anomenarem imatge de fons o model de fons.

La imatge de fons ha de ser una representació de l'escena sense objectes en moviment i ha de ser actualitzada contínuament per tal d'adaptar-se a possibles variacions de lluminositat. La literatura presenta molts mètodes per realitzar la segmentació de primer pla, un mètode senzill i eficaç és el *Running Gaussian average* [3].

Figura 4.1: Imatge original i segmentació de primer pla

## 4.1 Running Gaussian average

El mètode *Running Gaussian average* es caracteritza per modelar cada píxel independentment. Idealment cada píxel es caracteritzaria per una funció de densitat de probabilitat gaussiana, caracteritzada pels últims $n$ valors del píxel per cada color. Per tal d'evitar conservar n mostres per cada píxel i reduir el cost computacional, el model s'actualitza cada *frame* de la següent forma:

$$\mu_t^R = \alpha I_t^R + (1-\alpha)\mu_{t-1}^R \tag{4.1}$$

$$\mu_t^G = \alpha I_t^G + (1-\alpha)\mu_{t-1}^G \tag{4.2}$$

$$\mu_t^B = \alpha I_t^B + (1-\alpha)\mu_{t-1}^B \tag{4.3}$$

on $I_t^i$ és el valor actual del color vermell, verd o blau del píxel de la imatge actual, $\mu_t^i$ la mitja del model gaussià del píxel i $\alpha$ la taxa d'actualització. La taxa d'actualització marca un compromís entre l'estabilitat i una ràpida adaptació a noves condicions. La desviació típica $\sigma_t$ es pot calcular de forma similar, actualitzant el valor anterior segons la desviació de l'instant actual:

$$\sigma_t = \alpha D_t + (1-\alpha)\sigma_{t-1} \tag{4.4}$$

on $D_t$ és la diferència de color entre el píxel actual i el píxel del model:

$$D_t = \sqrt{(\mu_t^R - I_t^R)^2 + (\mu_t^G - I_t^G)^2 + (\mu_t^B - I_t^B)^2} = distancia(\boldsymbol{\mu_t}, \mathbf{I_t}) \tag{4.5}$$

Per tal de crear el model, es necessiten un nombre d'imatges d'entrenament a partir de les quals coneixerem l'estadística del fons per cada píxel. Un cop creat el model, classificarem un píxel com a primer pla si es compleix la inequació:

$$|D_t| > k\sigma_t \tag{4.6}$$

En cas contrari el classificarem com a fons i actualitzarem el model segons l'equació 4.1. D'aquesta forma només actualitzem el model si el píxel és caracteritzat com a fons.

A la figura 4.1 es pot observar el resultat de segmentar el primer pla d'una imatge. Com es pot observar, la segmentació delimita la zona de primer pla però presenta una sèrie de problemes: no es reconeix part del tamboret, ja que el seu color és molt semblant al fons, alguns píxels són marcats com a primer pla tot i no estar ocupats i s'observa un tall en la silueta resultant de la segmentació degut a la unió de la paret amb el terra. Es pot intentar reduir aquests tipus de problemes mitjançant diferents tècniques:

- *No deteccions de zones de primer pla* És un problema crític, ja que la reconstrucció es basa en les siluetes: si hi ha objectes que no estan recollits com a primer pla no es reconstruiran correctament. La solució és posar les condicions de detecció poc exigents ja que és preferible tenir falses deteccions en píxels de fons a no detectar algun objecte de primer pla.

- *Reflexions d'altres objectes* Els reflexos provoquen zones de falses deteccions. Per tal de reduir-los, abans de caracteritzar un píxel com a primer pla es comprova mitjançant uns llindars de distorsió de color i de luminància que efectivament la detecció de primer pla és correcta i no que vingui donada per una reflexió o ombra d'algun objecte proper.

- *Discontinuïtats en la segmentació* Les discontinuïtats poden ser degudes a zones de fons amb una gran variància, zones de l'objecte de primer pla que es confonen amb el fons o les zones de transició entre l'objecte i el fons. En tot cas, és important reduir-les ja que provocaran pèrdues (forats, talls) en la posterior reconstrucció. Amb l'objectiu d'eliminar aquest efecte, es pot realitzar una etapa de post-processament de la imatge de primer pla estimada basada en morfologia matemàtica.

## 4.2 Morfologia matemàtica

La morfologia matemàtica és una tècnica no lineal de processament d'imatge basada en operacions de conjunts. Es basa en comparar una imatge amb un element estructurant d'una forma preestablerta que es desplaça per sobre la imatge. Les seves característiques no lineals permeten combatre els errors de discontinuïtat de la segmentació presentats en el punt anterior.

Les dues operacions bàsiques de la morfologia matemàtica consisteixen en la dilatació i l'erosió:

### 4.2.1 Dilatació

El resultat de la dilatació és el conjunt de punts origen de l'element estructurant tals que l'element estructurant conté algun element del conjunt, quan l'element es desplaça per l'espai que conté els dos conjunts.

### 4.2.2 Erosió

L'erosió és el resultat de comprovar si l'element estructurant està completament inclòs dins del conjunt al que pertany el centre de l'element. L'efecte de realitzar una erosió és fer desaparèixer les estructures més petites i aprimar les majors.

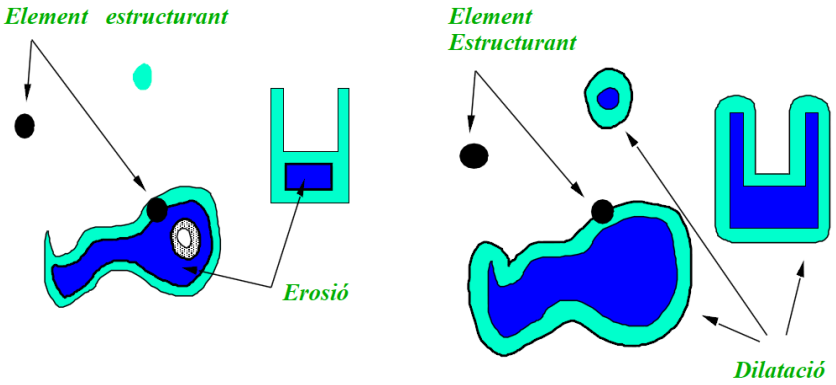


Figura 4.2: Resultat de realitzar un procés d'erosió i un de dilatació[4]

### 4.2.3 Tancament

El tancament és el resultat d'aplicar primer una dilatació i posteriorment una erosió amb el mateix element estructurant. El tancament evita petits forats dins l'objecte, omple parcialment els entrants i tanca els talls. Amb aquesta operació es podran tancar discontinuïtats en les siluetes.

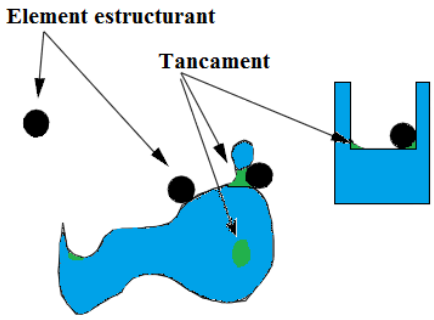


Figura 4.3: Resultat de realitzar un procés de tancament[4]

A la figura 4.2 es pot observar el resultat de realitzar una erosió i una dilatació. L'element estructurant és una esfera negra en els dos casos. A l'erosió el color blanc sobre fons negre es propaga al seu veïnat. Com a resultat, les franges

marcades amb turquesa passen a formar part del fons. A la dilatació es produeix l'efecte contrari, el fons negre es propaga a zones amb color blanc, donant com a resultat un objecte major (àrees de color turquesa).

Pel que fa a la dilatació observem que els objectes originals en blau són dilatats obtenint com a resultat la suma de les zones blaves i turqueses. Pel que fa al resultat del tancament es pot observar a la figura 4.3. Els objectes originals s'observen en blau mentre que el resultat del tancament seria la zona conjunta amb blau i verd.

Aquests exemples han estat extret del curs de morfologia matemàtica de Jean Serra [4].

## 4.3 Implementació

### 4.3.1 Organització de les dades

L'algorisme de segmentació de primer pla implementat no representa un coll d'ampolla per la seva càrrega computacional, però ens permet validar l'estratègia d'implementació paral·lelitzada per explotar correctament la *GPU*. En aquest punt es comenta la versió definitiva, deixant per l'apartat de resultats i anàlisi altres consideracions.

En la implementació realitzada es crea un *work-item* per cada píxel de la imatge. L'organització de les dades es realitza utilitzant estructures específiques per a imatges (*image_2d*) per a les imatges RGB d'entrada i les imatges d'escala de grisos de sortida. El model de fons utilitza un *buffer* de *cl_float4* pel valor mitjà de cada color i un *buffer* de *cl_float* per la variància.

Per tal de carregar les imatges a la *GPU* és necessari reorganitzar les dades prèviament ja que les dades en *CPU* utilitzen canals diferenciats per cada color mentre que les imatges a OpenCL utilitzen canals entrellaçats. La diferenciació en canals separats per cada color es tracta d'una decisió d'implementació de la llibreria del Grup d'Imatge.

De la mateixa manera que en el model de càmera, s'ha utilitzat una estructura de dades per disposar dels paràmetres a *GPU*.

### 4.3.2 Algorisme Running Gaussian Average

La implementació de l'algorisme *Running Gaussian Average* es realitza seguint el diagrama mostrat a la figura 4.4. Si es tracta d'un *frame* d'entrenament l'algorisme es limita a actualitzar el model. En cas de tractar-se d'un *frame* a segmentar es calcula la distància de color. Si supera la distància de color màxima i els llindars no detecten que es tracta d'una reflexió, el píxel serà marcat com a primer pla. Altrament es considerarà píxel de fons.

Si la distància de color indica que el píxel forma part del fons, s'actualitzarà el model. Aquestes actualitzacions permeten poder adaptar-se a canvis graduals d'il·luminació a l'escena.

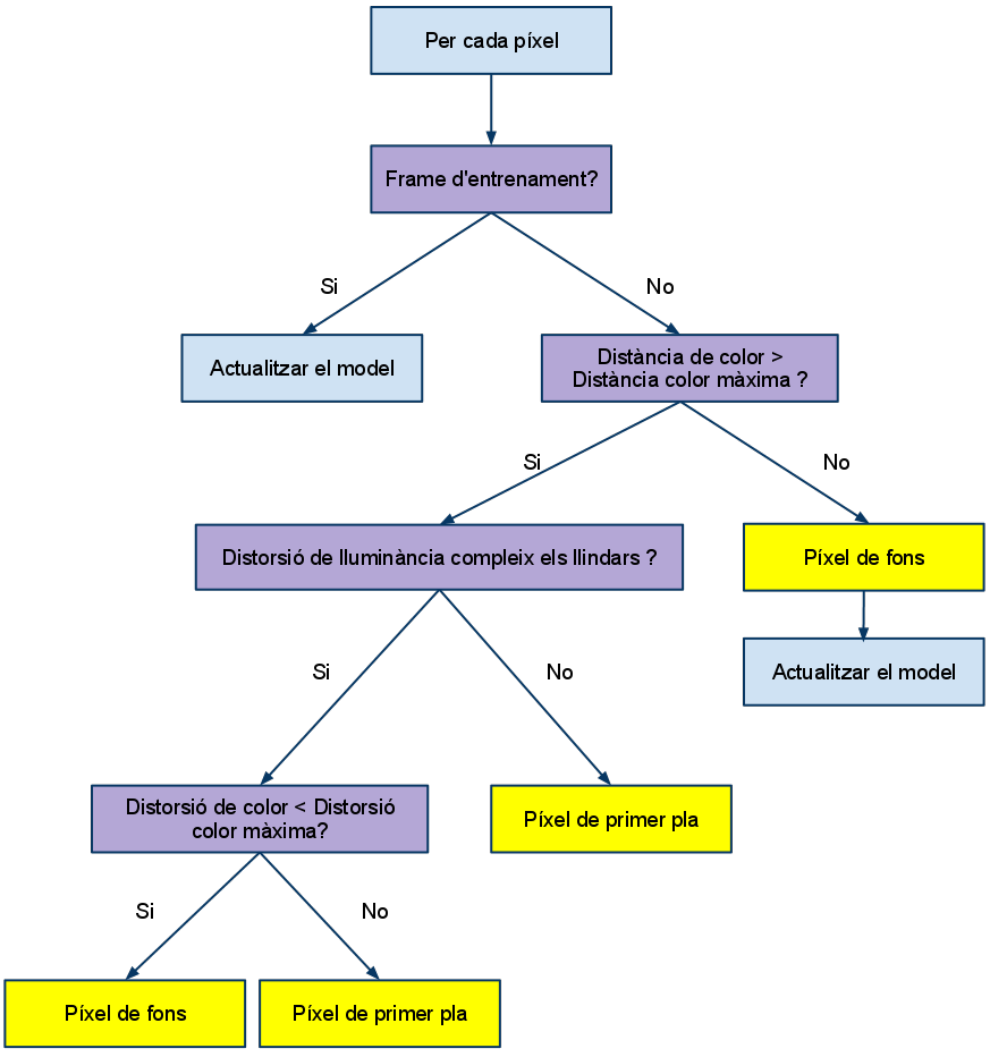


Figura 4.4: Diagrama de la segmentació de primer pla

La distància de color, distorsió de color i la distorsió de luminància es calculen:

$$color\_distance = |\boldsymbol{I_t} - \boldsymbol{\mu_t}| = ((\boldsymbol{I_t} - \boldsymbol{\mu_t})\dot{(}\boldsymbol{I_t} - \boldsymbol{\mu_t}))$$

$$brightness\_distortion = \frac{|\boldsymbol{I_t} \cdot \boldsymbol{\mu_t}|}{|\boldsymbol{\mu_t} \cdot \boldsymbol{\mu_t}|}$$

$$color\_distortion = |(\mathbf{I_t} - brightness\_distortion \cdot \boldsymbol{\mu_t}|$$

On $I_t$ és el valor actual del píxel RGB i $\boldsymbol{\mu_t}$ és el valor mitjà del model.

Els llindars de luminància i distorsió de color ens permeten detectar reflexions a l'escena, descartant alguns punts que d'altra manera es detectarien incorrectament com a primer pla.

### 4.3.3 Morfologia matemàtica

En el context d'aquest projecte, l'ús de la morfologia matemàtica es limita a millorar la segmentació de primer pla obtinguda. Per aquest objectiu amb una implementació binària de morfologia hagués estat suficient, però s'ha optat per una implementació més genèrica.

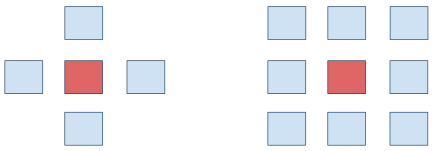

Figura 4.5: Elements estructurants en creu i en quadrat. El valor del píxel central és actualitzat en funció del seu valor i del valor dels píxels veïns

Els elements estructurants implementats s'observen a la figura 4.5. Es tracta d'elements estructurants simples que només recorren el píxel central i els 4 o 8 píxels veïns. Per utilitzar elements estructurants majors, les operacions de dilatació i erosió poden ser cridades repetidament.

La implementació realitzada consisteix en recórrer, per cada píxel de la imatge, tots els píxels de l'element estructurant. Actualitzant el valor central de l'element estructurant pel màxim de tots ells si es tracta d'una dilatació o del mínim en el cas d'una erosió.

Figura 4.6: Segmentació de primer pla sense tancament

A la figura 4.6 es mostra una imatge segmentada abans de realitzar el tancament. Es pot observar un tall a la segmentació corresponent a la cantonada d'una pissarra i problemes en la detecció de la mà. A la figura 4.7 es pot observar

Figura 4.7: Resultat d'aplicar tancaments de dimensió creixent

el resultat d'aplicar tancaments d'ordre 1, 4 i 5. A la imatge de la dreta no s'aconsegueix tancar el tall, a les altres dues el color blanc de la silueta es propaga al seu veïnat, tancant els talls.

En aplicar tancaments majors s'observa com es perd definició en les figures ja que les siluetes prenen forma de l'element estructurant. Així doncs és necessari trobar un compromís entre tancar correctament els talls en les siluetes sense perdre molta definició.

Amb els mètodes presentats fins aquest punt es possibilita la generació d'imatges sense distorsió i siluetes de primer pla amb una qualitat suficient per realitzar una reconstrucció multicàmera. Els algorismes presentats són fàcilment implementables en la *GPU*, degut a la possibilitat de processar cada píxel independentment tant pel que fa a la desdistorsió, la segmentació o el tancament.

# Capítol 5

# Reconstrucció tridimensional de l'escena

Existeixen diferents aproximacions al problema de reconstruir una escena a partir de múltiples vistes depenent del tipus de dades usades. En el present capítol es presenten dues aproximacions al problema, basades en segmentacions de siluetes per realitzar la reconstrucció.

## 5.1 Visual Hull

El concepte de *Visual Hull* va ser introduït per Laurentini [5] i es defineix com el màxim volum consistent amb les siluetes d'entrada, dit d'altra forma el *Visual Hull* d'un objecte és el volum més gran que projectat a les múltiples vistes produeix les mateixes siluetes. Les dades d'entrada són únicament les siluetes de les diferents vistes.

Hi ha diferents aproximacions al calcular el *Visual Hull* d'una escena, depenent de la unitat bàsica de representació elegida i de si es tracten les siluetes prèviament o si s'intenten millorar durant la reconstrucció, per tal de reduir el soroll. En aquest text es presenta únicament el mètode *Voxelized Shape from Silhouette*, que considera un processament previ de les siluetes i realitza la cerca de volums amb una voxelització regular.

Tractar les siluetes prèviament implica que no serà possible reduir el soroll present en elles amb la informació de les altres vistes, però en canvi, podran ser

tractades independentment limitant el cost i la complexitat de l'algorisme.

La voxelització regular es caracteritza per discretitzar l'espai en una graella 3D com a base per reconstruir l'escena. Cada punt d'aquesta graella es projecta a les diferents vistes, determinant a quin píxel de la imatge correspon. Si el píxel forma part de la silueta la càmera veurà el vòxel com a ocupat. Considerem que un vòxel forma part del *Visual Hull* si totes les càmeres el veuen com a ocupat.

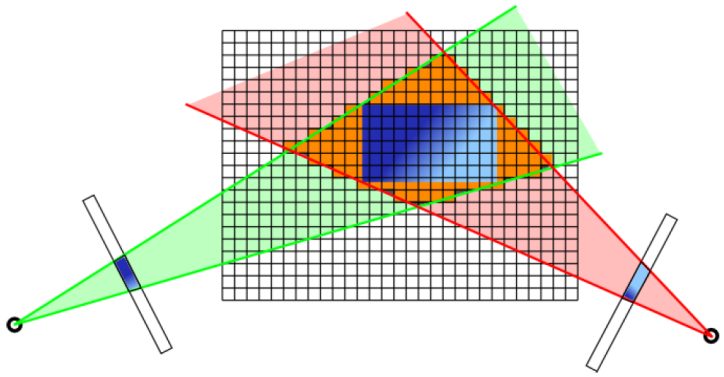

Figura 5.1: *Visual Hull* voxelitzat

La figura 5.1 mostra el *Visual Hull* voxelitzat amb dues càmeres d'un objecte quadrat de color blau. El *Visual Hull* resultant (vòxels marcats de color taronja a més de l'objecte en sí) ens mostra que, de les dues vistes disponibles, veuríem la mateixa silueta si l'objecte ocupés el volum blau que si ocupés el volum taronja.

## 5.2 Surface Sampling

El següent mètode presentat forma part de la tesi d'en Jordi Salvador 'Surface Reconstruction for Multi-View Video'[7]. L'objectiu del present projecte final de carrera és implementar aquest algorisme a *GPU*. A diferència del *Visual Hull*, s'allunya de les reconstruccions de volums per centrar-se en les superfícies dels objectes.

Les característiques rellevants en entorns multicàmera es troben a les superfícies dels objectes, ja que l'interior dels quals queda ocult per la superfície opaca. Per tant, la zona d'interès en les reconstruccions d'escenes 3D es limita a la interfície entre els objectes opacs i l'espai transparent al seu voltant. Amb la concentració dels punts de representació a la superfície dels objectes s'aconsegueix millorar la reconstrucció 3D amb un menor suport de dades en front una voxelització regular.

Aquesta superfície pot ser representada ja sigui per punts de superfície aïllats o bé per una malla poligonal. Treballar amb punts de superfície permet assolir un nivell de detall a les superfícies augmentant la densitat en la regió d'interès. La representació en malles permet l'estimació de la superfície entre mostres existents, mecanisme molt utilitzat en acceleració de gràfics.

Una altra possibilitat és treballar amb punts de superfície amb una orientació associada, elements anomenats *surfels*. L'orientació dels *surfels* permet disposar d'informació local de la superfície, útil per trobar nous punts de superfície propers o per realitzar un post-processat. És per aquest motiu que seran utilitzats en la reconstrucció.

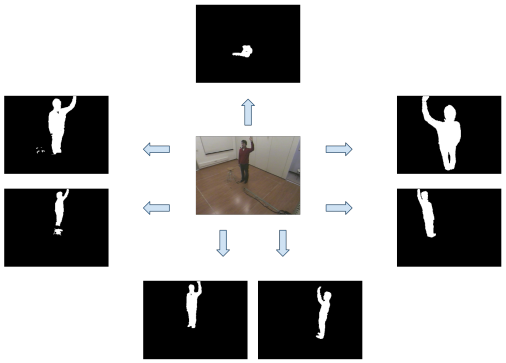

Figura 5.2: Segmentació d'una escena per les diferents vistes. *Visual Hull* i *Surface Sampling* es basen en la segmentació de primer pla per realitzar la segmentació.

### 5.2.1 Cerca de punts de superfície

L'algorisme busca *surfels* a partir d'un mostreig aleatori de punts de l'espai. Com els mètodes anteriors, utilitza siluetes per tal de realitzar la reconstrucció. Per tal de considerar que un punt de l'espai és un *surfel*, és necessari que es compleixin les següents condicions:

- La projecció del punt a cada vista forma part del primer pla. Estrictament un punt de superfície ha de formar part del primer pla de totes les vistes però es pot considerar una tolerància per robustesa en front a errors de segmentació. Superar aquesta condició implica que el punt forma part del *Visual Hull* de l'escena

- La projecció forma part d'un píxel de contorn en alguna vista. Si un punt de l'espai es projectat en un píxel $\mathbf{p}_c$, aquest píxel es considerat de contorn

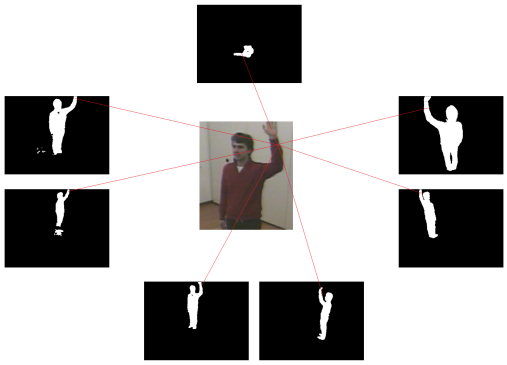

Figura 5.3: Projecció d'un punt a les diverses siluetes.

si forma part del primer pla i algun dels quatre píxels adjacents forma part del fons. D'aquesta forma ens assegurem que el punt que formava part del *Visual Hull* per la primera condició és un punt de superfície

- Un *surfel* queda determinat per un punt i una orientació. Per tant, ha de ser possible estimar l'orientació del punt a partir d'un veïnat proper. Un cop trobat un punt es busca en un veïnat proper punts de superfície que serviran per determinar-ne l'orientació

- Es calcula l'orientació del *surfel* a partir del punt i els seus veïns

### Estratègia de cerca

Les condicions anteriors permeten determinar el criteri per considerar que un punt de l'espai és un punt de superfície. La generació dels punts candidats a esdevindre punts de superfície es pot fer seguint diferents estratègies. Una possible obtenció dels punts consistiria en la generació aleatòria de punts en tot l'espai de cerca. Es tracta d'una cerca poc eficient ja que la probabilitat de trobar un punt de superfície en un volum sense cap informació prèvia és baixa. Amb una bona estratègia de cerca es pot augmentar l'eficiència de cerca de punts i el rendiment de l'algorisme.

A continuació es presenten dues possibles estratègies de cerca. En la primera estratègia -inicialització- es parteix únicament d'un joc de siluetes mentre que a la segona -propagació- s'aprofita, a més a més, informació d'una reconstrucció anterior.

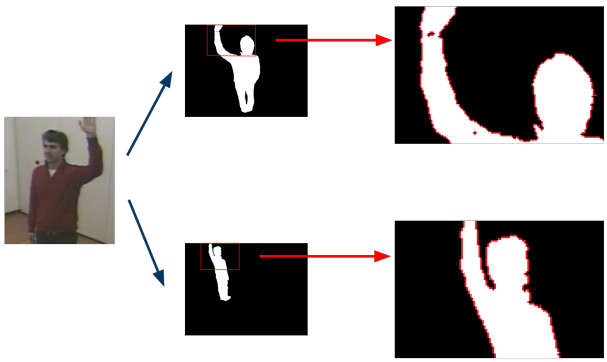

Figura 5.4: Ampliació de les siluetes. Els punts que són projectats als píxels de color vermell són detectats com a punts de superfície per aquella càmera.

### Inicialització

L'algorisme d'inicialització proposa la reconstrucció de superfícies d'una escena a partir d'un joc de siluetes. Aprofita el progressiu coneixement de l'escena per augmentar l'eficiència de cerca de punts, garantint-ne una distribució densa i uniforme per tota la superfície. Consisteix en dues etapes: una primera de cerca de llavors i una posterior d'expansió.

La primera etapa s'encarrega de buscar *surfels* per tot el volum de treball. L'objectiu d'aquesta etapa és aconseguir un conjunt de punts de superfície als que anomenarem llavors. Aquest procés requereix un gran nombre d'intents per tal de trobar cada llavor ja que la probabilitat de trobar un punt de superfície en un volum és baixa.

Un cop trobat el nombre desitjat de llavors en la primera etapa, es passa a la fase d'expansió. Durant la fase de cerca de llavors s'han trobat un nombre de *surfels* repartits aleatòriament per la superfície. La fase d'expansió extreu noves mostres de superfície en un veïnat proper a una llavor ja obtinguda. El resultat del procés d'expansió és una reconstrucció més densa. L'objectiu és utilitzar les mostres de superfície ja trobades per reduir el nombre d'intents abans de trobar nous punts de superfície. L'expansió es realitza a partir d'una llavor i d'un desplaçament aleatori en una zona propera.

L'expansió pot ser realitzada amb diverses regions d'expansió. Una possible regió de cerca consisteix en una esfera, el centre de la qual es troba en una llavor. Es tracta d'una expansió omnidireccional ja que no discrimina entre les tres coordenades de l'espai.

Una regió més elaborada consisteix en realitzar l'expansió utilitzant la informació d'orientació del punt. Amb la informació d'orientació de la llavor es pot ponderar el desplaçament en la direcció de l'orientació del punt en front dels altres dos eixos. La ponderació redueix l'espai de cerca en la direcció ortogonal a la superfície ja que s'espera que a la direcció normal no es trobi cap altra superfície del mateix volum. Diferenciant entre els tres eixos de l'espai generem una regió de cerca rectangular.

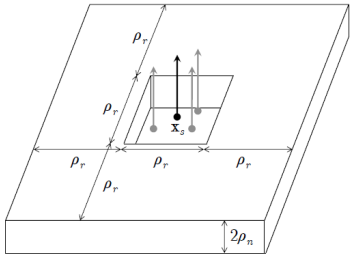


Figura 5.5: Regió d'expansió rectangular a partir d'un punt de superfície $\mathrm{x}_s$. Afavoreix l'expansió al pla ortogonal a la normal del punt, permetent petites variacions en la direcció de la normal

Per tal de no acumular moltes mostres en un espai reduït s'afegeix un desplaçament constant que assegura una dispersió entre la llavor i el nou punt trobat. La figura 5.5 mostra la regió de cerca definida.

La fase d'expansió consisteix en realitzar cerques de punts amb la regió descrita dimensions progressivament menors. Amb aquest procediment s'aconsegueix augmentar progressivament l'eficiència de cerca així com la densitat dels punts de superfície.

### Cerca Dinàmica

L'algorisme d'inicialització queda limitat per la fase de generació de llavors. Per tal d'aconseguir millors rendiments es pot utilitzar la informació d'una reconstrucció anterior per generar les llavors en l'instant actual. Es limita l'espai de cerca a una zona propera a on es trobaven els punts de superfície en l'instant anterior ja que l'escena no haurà variat excessivament d'un instant al següent al treballar amb un nombre elevat de reconstruccions per segon. Un cop trobades les llavors es podran expandir com en la segona part de l'algorisme d'inicialització.

L'algorisme de propagació parteix d'un nombre de mostres de superfície de l'instant anterior extretes aleatòriament de la col·lecció de punts de superfície. A

partir de cada punt de l'instant anterior realitza una cerca aleatòria en una regió esfèrica per tal de no afavorir cap direcció de l'espai tal com mostra la figura 5.6.

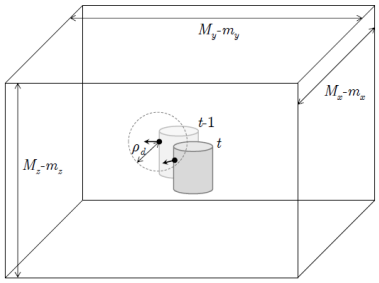


Figura 5.6: Aprofitant la correlació temporal entre *frames* consecutius es possible reduir l'espai de cerca de llavors a una esfera de radi $\rho_d$, incrementant l'eficiència de cerca

La propagació a partir d'una col·lecció de punts anterior aconsegueix una cerca de llavors molt més eficient però no garantitza una separació entre elles ja que l'espai de cerca està condicionat per mostres anteriors extretes aleatòriament. La fase d'expansió ha d'aconseguir una bona densitat per tota la superfície dels volums a reconstruir.

### 5.2.2 Post-processament

Un cop trobats el nombre de punts desitjat, es pot millorar el resultat final realitzant un millor càlcul de l'orientació de cada *surfel* així com un suavitzat dels punts de superfície. També es possible associar a cada punt un color per tal de poder representar les superfícies en color.

#### Càlcul de normals

En la cerca de punts s'estima l'orientació de cada *surfel* a partir d'un nombre de veïns. Aquesta estimació permet obtindre una normal a la superfície que és útil per la fase d'expansió. Aquesta estimació, però, es basa en pocs veïns i per tant és sorollosa. Per tal d'utilitzar l'orientació per suavitzar la superfície o acolorir els punts es requereix una millor orientació.

Un cop trobats tots els *surfels*, es busquen tots els veïns que es trobin a una distància menor que un cert radi i estimar una normal amb tots ells. Per tal d'evitar recórrer tots els punts en la cerca de veïns, els punts de superfície poden ser ordenats en una estructura de dades permetent una cerca més eficient. Una

estructura de dades útil per aquest propòsit és el *kd-tree* o arbre k-dimensional [9].

Tant per l'estimació de l'orientació en un veïnat local com l'estimació a partir de tots els punts, el mètode per estimar l'orientació dels punts és el mateix:

- Seleccionar els punts a treballar al veïnat:

  - En cas de treballar amb tots els veïns, es busca el conjunt de $v_i = \{\mathbf{x}_j\}$ amb tots els punt $x_j$ a una distància de $x_i$ igual o menor a $r_n$, on $r_n$ és la màxima distància per considerar dos punts veïns

  - En cas de treballar amb l'estimació local, tots els veïns locals $x_j$ formaran part del veïnat $v_i$

- Estimar la posició mitjana $\mu_i$ de $v_i$. Modificar cada punt $x_j$ extraient la posició mitjana: $x_j := x_j - \mu_i, \forall x_j \in v_i$

- Crear la matriu $\mathbf{M}_i = \sum_{x_j \in v} x_j x_j^T$

Un vector $\hat{\mathbf{n}}_i$, ortogonal al pla que minimitza l'error quadràtic mitjà al veïnat $v_i$, es obtingut com el vector propi de major valor propi de $\mathbf{M}_i$. El signe de $\hat{\mathbf{n}}_\mathbf{i}$ presenta un grau de llibertat que ha de ser determinat. Petits desplaçaments del punt en direcció a la normal permeten determinar si el signe és correcte. La projecció a les diverses vistes comprova si es correcte (el punt desplaçat es troba en espai buit) o incorrecte (el punt es troba dins d'un volum ocupat). El mètode de càlcul de normals es mostra a la figura 5.7.

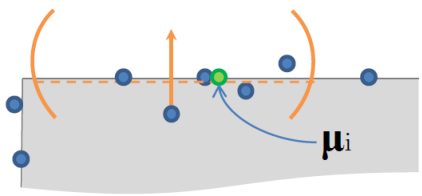


Figura 5.7: Normal estimada a partir d'un grup de punts. $\mu_i$ és el punt mitjà que resulta de promitjar el conjunt de punts $v_i$

### Suavitzat de superfície

El mostreig aleatori de l'espai i la precisió limitada de les siluetes utilitzades provoquen un soroll en la posició dels punts de superfície. Per tal de reduir

aquest soroll, es pot utilitzar el veïnat local i l'orientació del punt per realitzar un desplaçament de cada punt en la direcció de la normal. Aquest desplaçament permet alinear-lo amb els punts veïns mantenint una separació entre ells i evitant la formació de *clusters*. La figura 5.8 mostra el suavitzat de superfície amb el desplaçament limitat en la direcció de la normal.

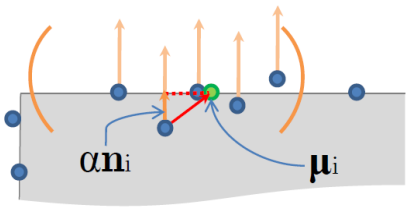


Figura 5.8: Suavitzat de la superfície, limitant el desplaçament de cada punt en la direcció de la normal $v_i$

- Seleccionar els punts a treballar al veïnat:

  - Es busquen el conjunt de $v_i = \{\mathbf{x}_j\}$ amb tots els punt $x_j$ a una distància de $x_i$ igual o menor a $r_n$. Descartarem aquells punts la normal dels quals difereixi de la normal del punt un cert angle $\hat{n}_i \cdot \hat{n}_j > \theta_v$

- Estimar la posició mitjana $\mu_i$ de $v_i$. Modificar el nou punt $x_i^o$ en la direcció de la normal : $x_i^o := x_i + \alpha\hat{n}_i$, amb $\alpha = (\mu_i - x_i) \cdot \hat{n}_i$

- Si la projecció de $x_i^o$ no cau a cap píxel de superfície, repetir el pas anterior fitxant $\alpha := \frac{4}{5}\alpha$

### Acolorir els punts de superfície

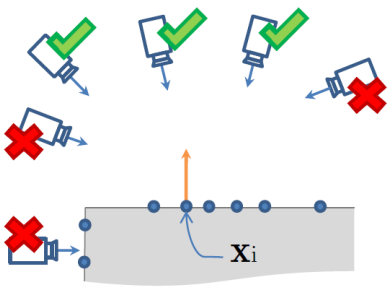


Figura 5.9: En el procés d'acolorir un punt es descarten les càmeres que no passen el test de visibilitat, s'utilitza la informació de les càmeres més ben alineades (vectors blaus) amb l'orientació del punt (vector taronja) $v_i$

Per tal d'afegir la informació de color en cada punt és necessari comprovar prèviament la visibilitat del punt per part de les diverses càmeres. Un punt serà visible per una càmera si té línea de visió directa amb el centre de projecció de la càmera. O el que és el mateix, el punt forma part de l'àrea de visió de la càmera i no hi ha cap volum que s'interposi entre la càmera i el punt.

La reconstrucció de superfície presentada treballa amb punts de superfície aïllats, sense coneixement de les característiques globals de la superfície. Per aconseguir la visibilitat de cada punt a les diverses càmeres és necessari construir un mapa de profunditat per a cada càmera previ a l'assignació de color a cada punt. El mapa de profunditat d'una imatge indica la distància de cada píxel a les superfícies dels objectes de l'escena des d'un punt de vista.

Un cop calculat els mapes de profunditat, cada punt pot ser acolorit amb les 3 càmeres que, veient el punt, es trobin més ben orientades com indica la figura 5.9

## 5.3 Implementació

### 5.3.1 Problemàtica de la paral·lelització

Una de les dificultats a l'hora de realitzar la paral·lelització d'un algorisme és la forma de repartir la càrrega de treball entre els diferents *work-item* d'execució. Per tal d'il·lustrar aquesta problemàtica s'exposen diferents possibilitats per la implementació del mètode que ens ocupa. Es pren com a referència la primera part de l'algorisme de cerca de punts, concretament la part encarregada de la cerca de punts i el càlcul de les normals.

El procés de cerca de punts i les normals associades es pot dividir en tres blocs diferenciats. Un primer bloc cerca els punts de superfície, ja siguin buscats aleatòriament per tota la zona de treball o per una zona més reduïda. Un segon bloc correspon a la cerca dels veïns per cada punt de superfície en un espai de cerca acotat. Finalment, el tercer bloc correspon al càlcul de les normals per a cada agrupació de punt-veïns.

Tenint presents aquests blocs d'alt nivell, l'execució es pot organitzar en:

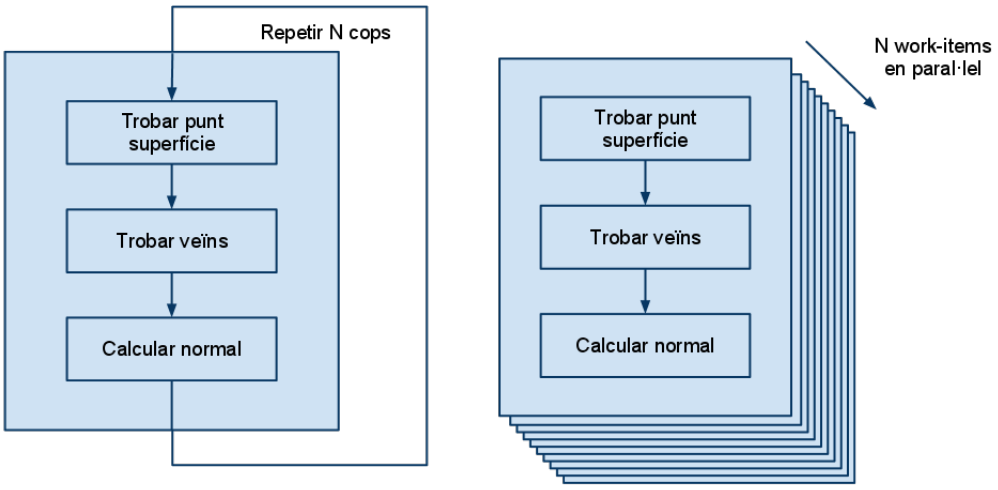


Figura 5.10: Execució seqüencial de l'algorisme comparada amb execució en paral·lel

- Implementació seqüencial: el mètode busca un punt, seguidament busca els veïns i calcula la seva orientació. Realitza aquest seguit d'operacions fins a assolir un nombre de punts determinat. Un sol fil d'execució realitza totes les operacions N cops

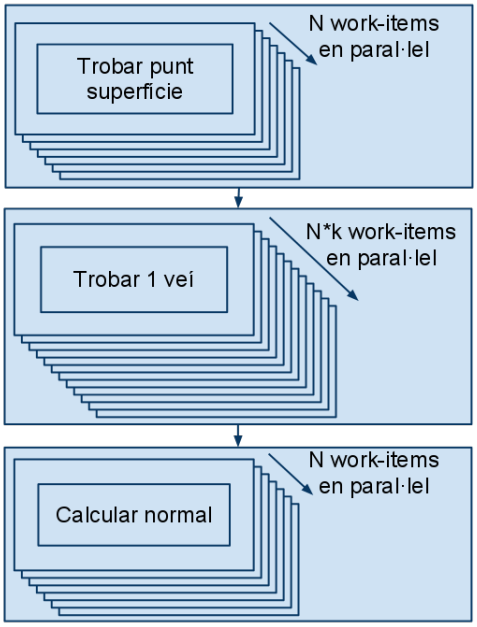


Figura 5.11: Execució en paral·lel de l'algorisme en diferents etapes. Es creen diferent nombre de *work-item* segons l'etapa de l'algorisme

- Càlcul paral·lelitzat de cada agrupació punt-veïns: cada fil d'execució s'encarrega de buscar un punt de superfície, els seus veïns i calcula les normals. Els *work-item* no acaben fins trobar un punt i els seus veïns. La figura 5.10 mostra la comparació entre una execució seqüencial amb cerca de N punts i l'execució en paral·lel

- Càlcul paral·lelitzat amb diferenciació de tasques: cada etapa de l'algorisme és calculada separadament. En primer lloc es creen N *work-item* que s'encarreguen de cercar els punts de superfície. Un cop han estat trobats, la segona etapa cerca els veïns, creant un fil per cada veí a trobar. Finalment N *work-item* són creats per calcular les normals. Diferenciant entre tasques, es poden crear diferent nombre de *work-item* a cada etapa. La figura 5.11 mostra aquesta execució paral·lela amb diferents etapes

- Cada configuració paral·lela permet l'organització dels *work-item* en una, dues o tres dimensions. Conceptualment l'execució és la mateixa però variant l'estructura dels *work-item* el rendiment pot diferir ja sigui per qüestions de hardware, d'accessos a memòria o per la mida del *work-group* en front la mida global. La figura 5.12 mostra dues possibilitats per a l'ordenació dels *work-item*

Les diferents versions enunciades assoleixen resultats d'igual qualitat, però

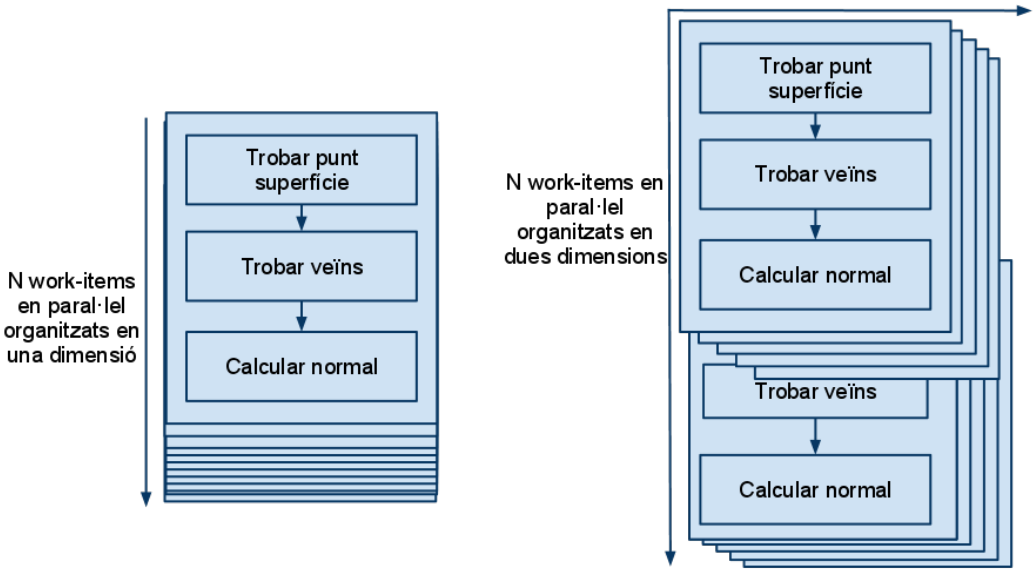


Figura 5.12: Execució en paral·lel de l'algorisme amb organitzacions dels fils d'execució en una i dues dimensions

amb temps d'execució molt diferents. És necessari trobar les configuracions amb millor rendiment. El capítol 6 analitza les diferents possibilitats explorades. En aquest capítol es proposa la solució adoptada.

### 5.3.2 Solució proposada

El mètode de reconstrucció de superfícies presentat cerca punts de superfície aïllat, buscats independentment d'altres punts. Diverses etapes utilitzen informació de punts ja trobats per augmentar l'eficiència de cerca.

La cerca independent de punts fa pensar en que es tracta d'un algorisme ideal per ser paral·lelitzat ja que es poden executar molts *work-items* en paral·lel on cadascun cercarà punts de superfície. Un cop paral·lelitzat l'algorisme s'observa que el rendiment obtingut no només no millora la implementació seqüencial, si no que l'execució és clarament més lenta.

Aquest comportament és degut a que el mètode de reconstrucció de superfícies presentat es basa en realitzar múltiples projeccions a les diverses càmeres per tal de detectar punts de superfície. Aquestes projeccions comproven si la projecció a cada imatge forma part de la silueta, sent necessari recórrer el píxel on es projecta el punt i els seus veïns.

La gran capacitat de les targetes gràfiques provoca que per alguns algorismes, com en aquest cas, el rendiment estigui limitat per la velocitat dels accessos a memòria i no per la capacitat de càlcul. Els múltiples processadors requereixen llegir dades d'una mateixa memòria, augmentant la latència dels accessos.

Les *GPUs* emmascaren la latència d'aquests accessos processant altres *work-items*. Si l'algorisme requereix de múltiples accessos a memòria consecutius, els múltiples *work-items* que processa una *Compute Unit* requeriran de dades per poder seguir l'execució.

Per tal de millorar el rendiment de l'algorisme és necessari reduir el nombre d'accessos a memòria. Per aquest fer es proposa realitzar un *visual hull* voxelitzat de l'escena previ a la cerca de punts. La voxelització permet acotar la zona de cerca a zones on la probabilitat de trobar punts de superfície és superior.

Es tracta d'una aproximació al problema diferent a l'execució seqüencial. El llistat d'operacions a realitzar és el següent:

- Transferir el joc d'imatges de segmentació de primer pla a la *GPU*
- Realitzar el *Visual Hull* voxelitzat de la zona d'interès
- Detectar les regions de superfície (Bins de superfície): una cerca als vòxels adjacents permet detectar les regions amb alta probabilitat de trobar punts de superfície
- A partir dels bins es busquen punts de superfície en una regió acotada
- La cerca de veïns necessaris per estimar l'orientació del punt es realitza a partir dels punts de superfície ja trobats
- Es realitza el suavitzat de superfície a partir de l'orientació del punt
- Finalment s'assigna color als punts de superfície a partir de la seva posició i l'orientació. Es tracta d'una operació que pot ser executada posteriorment a la cerca de punts de superfície ja que utilitza les imatges RGB originals com a dades

Un cop presentada l'estructura de l'algorisme es detallen les característiques de cada etapa:

Figura 5.13: Resultat de la voxelització

### Determinar els bins de superfície

L'algorisme seqüencial aprofita el coneixement progressiu de punts de superfície per millorar l'eficiència de cerca. Després d'explorar diferents possibilitats sense èxit, s'abandona l'aproximació seqüencial d'augmentar progressivament l'eficiència de cerca a través de diverses expansions, substituint-ho per un *visual hull* previ a la cerca de punts de superfície.

Amb la implementació proposada es mescla el mostreig regular usat en les voxelitzacions amb el mostreig irregular de la cerca de punts de superfície. Amb la voxelització s'aconsegueix detectar les zones on es trobaran els punts de superfície, aconseguint una representació de baixa resolució amb baix cost computacional. Amb la cerca de punts aleatoris de superfície s'aconsegueix augmentar arbitràriament la resolució, amb l'avantatge que l'espai de cerca és reduït.

El *visual hull* permet aconseguir informació de la posició dels objectes de l'escena evitant realitzar tants accessos a memòria com la cerca de punts de superfície. El seu objectiu és buscar punts de l'espai ocupats, en contraposició als punts de superfície que, a més d'estar ocupats, requereixen algun punt proper de l'espai buit.

Al centrar-se en la cerca de punts ocupats el nombre de projeccions requerides és menor ja que amb un accés al píxel projectat a cada càmera es suficient, mentre que en la cerca de punts de superfície es realitzen 5 accessos per cada càmera.

El mostreig regular de tot l'espai realitzat per la voxelització permet adquirir un bon coneixement de l'escena. L'estructura regular de la voxelització, a més a més, permet explorar els vòxels veïns per determinar si es troben o no ocupats, podent descartar els que es troben envoltats de vòxels ocupats ja que en ells no trobarem punts de superfície.

Un cop descartats els vòxels envoltats per vòxels també ocupats es disposa d'una col·lecció de regions de l'espai amb alta probabilitat de trobar punts de superfície. Per tal de no confondre termes, es referirà al resultat de la voxelització com a bins en lloc de vòxels. Aquesta nova notació fa palès que la voxelització és un pas intermedi de l'algorisme (les 'caixes' o 'regions' on es buscaran els punts de superfície) i no un resultat.

### Cerca de punts de superfície

A partir dels bins de superfície es busquen punts de superfície en una esfera de radi la separació entre bins. La cerca en la esfera és aleatòria, sense cap ponderació en les diverses direccions de l'espai.

El radi elegit possibilita una superposició parcial entre els diversos bins. Aquesta superposició evita àrees de superfície amb menor densitat de punts a les zones d'unió entre bins. També ofereix una redundància que permet evitar errors en el mostreig regular.

El nombre de *work-items* de cerca es fixa abans de l'execució. Els *work-items* es reparteixen automàticament entre els diferents bins ja que en cada *frame* el nombre de bins serà diferent.

Cada *work-item* té un nombre d'intents màxim per trobar un punt de superfície. Si no l'ha trobat en aquest nombre d'intents desisteix. La justificació a aquesta limitació es troba al capítol següent. El nombre de punts de superfície trobats a cada instant de temps variarà.

La selecció en bins permet que amb un nombre reduït d'intents un gran percentatge dels *work-items* trobin punts de superfície. Quan un *work-item* troba un punt de superfície, guarda el número de bin al qual pertany. Cada punt es trobarà associat al bin on ha estat trobat per tal de facilitar el posterior càlcul de normals.

A diferència de la implementació seqüencial, un cop trobat un punt de superfície no es busquen veïns i normals ja que els punts no són expandits i no es

necessita una informació de l'orientació en aquesta etapa de l'algorisme.

### Càlcul de l'orientació dels punts

Un cop trobats tots els punts de superfície, es calcula l'orientació de cada punt. Aprofitant la distribució en bins es busca, per cada punt, tots els veïns a una certa distància. Sense la distribució en bins, una cerca per tots els punts de superfície trobats no seria viable.

L'orientació del punt s'estima a partir del punt i dels veïns. Aquesta orientació usualment es recolza en més punts que el nombre de veïns buscats pel mètode seqüencial, podent reduir l'error en l'estimació.

El solapament parcial dels bins provoca que alguns punts de superfície propers al punt no siguin considerats en la cerca, ja que formen part del bin contigu. Aquest fet pot comportar divergències en l'estimació de la normal entre punts propers.

Per tal de calcular l'orientació dels punts es creen tants *work-items* com en la cerca de punts de superfície. Alguns *work-items* no realitzaran cap treball ja que en la cerca de punts no tots els *work-items* hauran trobat un punt.

### Suavitzat de superfície

Un cop calculada l'orientació de cada punt es pot realitzar el suavitzat de superfície. Es tornen a buscar tots els veïns a una distància menor a un llindar, si es desitja diferent del llindar utilitzat en el càlcul de normals.

Amb les normals calculades anteriorment es realitza un desplaçament dels punts en direcció a la normal en funció de la posició dels seus veïns. Novament l'organització en bins pot suposar una diferència de suavitzat entre punts de bins veïns. En aquest cas es creen també tants *work-items* com el nombre de punts cercat anteriorment.

### Acolorir els punts de superfície

Per tal de representar els punts de superfície és necessari acolorir-los utilitzant càmeres amb línea de visió als punts. L'acoloriment dels punts consta de dues

etapes: creació dels mapes de profunditat i assignar color als punts.

- Determinar el mapa de profunditat de cada càmera. Implementació amb *depth maps* d'OpenGL. Per cada càmera es realitzen els següents passos:

  Es configura OpenGL amb l'ús de depth maps, carregant el model de càmera i la posició on es troba

  Es dibuixen tots els punts de la reconstrucció com a vertex3D. No són visualitzats, però si que es modifica el mapa de profunditat

  Es llegeix el mapa de profunditat

- Cada punt es projectat a les diverses càmeres. Si el punt es troba a una distància menor a la indicada pel mapa de profunditat més una certa tolerància es compara l'orientació del punt amb l'orientació de la càmera. S'acoloreix el punt amb una ponderació de les tres càmeres amb millor orientació

En aquest capítol s'ha presentat el mètode de reconstrucció 3D de superfícies seqüencial. A partir del mètode seqüencial s'ha presentat una metodologia alternativa indicada per a l'execució eficient en *GPUs*, amb l'objectiu d'assolir temps real.

# Capítol 6

# Resultats i anàlisi

En aquest capítol es presenten els resultats obtinguts agafant com a referència una implementació dels algorismes seqüencials en *CPU*.

La *CPU* utilitzada per els càlculs de temps és Intel Core 2 Duo E7400 (2.80GHz). La targeta gràfica utilitzada pel codi OpenCL és una NVIDIA GeForce GTX 295. Es tracta d'un xip que conté dues *GPU* amb 240 nuclis cada una a 1.242 MHz. Per la implementació realitzada només s'ha utilitzat una de les dues *GPUs* del xip.

## 6.1 Projecció i model de càmera

El capítol Projecció i model de càmera presenta l'aproximació a dos problemes: la relació de punts de l'espai amb píxels de les càmeres per una banda i l'eliminació de la distorsió de lent. La projecció de punts 3D a imatges 2D és una eina per a la reconstrucció 3D i no una funcionalitat en si mateix. No es poden aconseguir resultats per la projecció, més enllà de comprovar que funciona correctament.

La compensació de la distorsió de lent si que pot ser analitzat com a resultat, si bé acostuma a realitzar-se conjuntament amb la segmentació de primer pla. Es tracta d'una operació altament paral·lelitzable, ja que cada píxel treballa independentment de la resta, de forma que s'aconsegueix una millora d'aproximadament 70 vegades en la implementació amb OpenCL.

La compensació de distorsió, en ser una de les primeres tasques a realitzar

va permetre provar diferents configuracions per tal de familiaritzar-se amb les possibilitats i les limitacions d'OpenCL.

En primer lloc cal destacar les dificultats inicials ja que el tractament d'errors a OpenCL 1.0 no es troba molt ben documentat. Els errors dins dels *kernels* són complicats de debugar ja que no es rep informació sobre ells, els errors al *host* program són més fàcils de detectar. També s'ha observat disparitat d'implementacions entre diversos fabricants (NVIDIA - ATI): la forma d'accedir als paràmetres té diferents possibilitats en una implementació que a l'altra, l'accés a matrius dins dels *kernels* també presenta incongruències. Es d'esperar que en properes versions d'OpenCL aquestes diferències vagin desapareixent.

Deixant de banda l'estat actual d'OpenCL, el treball sobre segmentació ha permès extreure les següents conclusions:

- En *kernels* amb poc treball sobre les dades, un gran percentatge del temps és ocupat per les transferències de memòria entre el *host* i el *device*. La duració d'aquestes transferències depenen de l'ample de banda de la connexió *host-device*, per tant l'únic que es pot fer per limitar-ho és transmetre les dades amb el mínim nombre de bits possible
- La gran capacitat de càlcul de la *GPU* utilitzada permet emmascarar el temps de transferència de dades
- Majors dimensions de *work-groups* assoleixen millors rendiments
- El rendiment usant buffers i el tipus específic per imatges d'openCL és molt semblant. El tipus específic d'OpenCL té accessos específics a les dades optimitzats per a l'arquitectura OpenCL

Figura 6.1: Imatge original i imatge desdistorsionada

El resultat obtingut en la implementació en OpenCL es troba desplaçat respecte a la implementació en *CPU*. Aquest desplaçament és de l'ordre d'un píxel,

probablement degut a l'efecte del sampler. Es tracta d'una diferència mínima que no ocasiona problemes posteriors.

Pel que fa als temps d'execució, la figura 6.2 mostra un gràfic amb els temps de la compensació de distorsió en *CPU* i *GPU*. La implementació en *CPU* és més lenta per qualsevol mida d'imatge, en augmentar la mida d'imatge la diferència de temps entre una implementació i l'altra augmenten. Al augmentar el nombre de dades a processar, milloren les prestacions de l'algorisme paral·leltitzat en front del sèrie. La temps de transferència de dades entre el *host* i el dispositiu queda emmascarat per la gran capacitat de càlcul de la *GPU*.

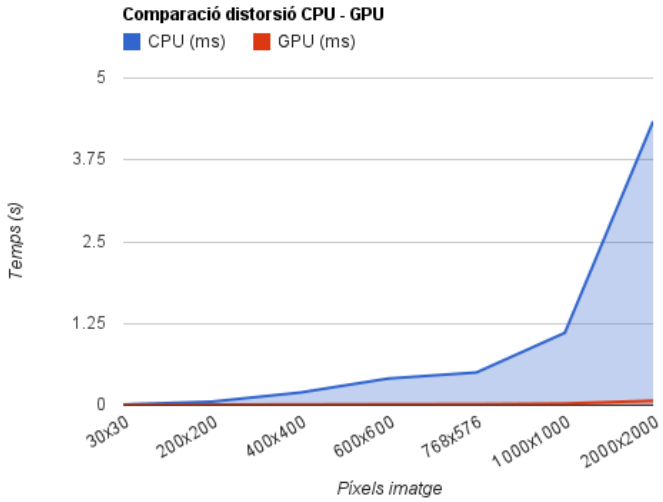


Figura 6.2: Comparativa temps d'execució *CPU* - *GPU*

La mida de les imatges captades per les càmeres utilitzades en aquest projecte és de 752 columnes per 582 files o de 376 columnes per 288 files si treballem en baixa resolució (veure capítol 7). En ambdós casos la implementació amb *GPU* introdueix una gran millora respecte a la versió en *CPU*.

## 6.2 Segmentació de primer pla

El comportament de la segmentació de primer pla implementada és semblant a la compensació de distorsió. Es tracta d'un algorisme menys costós computacionalment, de forma que les transferències de memòria tenen un major pes al temps d'execució total.

El rendiment de la segmentació en *GPU* torna a ser superior que en *CPU* tal com s'observa a la figura 6.3. Com en la compensació de distorsió, la millora de rendiment en paral·lelitzar es fa palès al augmentar les dimensions de la imatge a tractar.

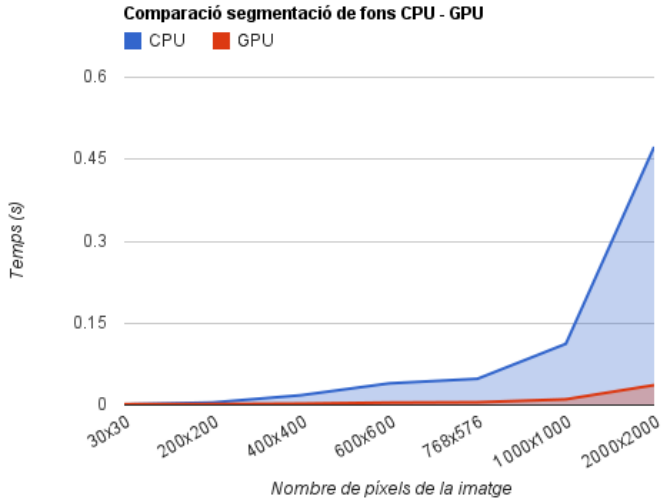


Figura 6.3: Comparativa temps d'execució *CPU - GPU*

Cal destacar que la versió d'OpenCL 1.0 d'NVIDIA no permetia l'ús d'imatges d'un sol canal mentre que la versió 1.1 ja ho suporta. Al extreure les siluetes només es necessita un canal i l'ús de 4 implicava un temps de transferència major i un posterior reordenament de les dades que feia l'ús de buffers lleugerament més interessant. Al disposar d'imatges d'un sol canal, el rendiment es superior al de buffers i per tant s'opta per aquesta opció.

La figura 6.3 mostra la comparació en temps d'execució entre la implementació seqüencial i la implementació paral·lela. En ambdós casos es realitza un tancament d'ordre 1 a la segmentació de primer pla resultant. La potència de paral·lelitzar l'algorisme s'observa en augmentar la mida de la imatge. Per imatges de mida molt reduït la implementació seqüencial és més ràpida, però amb les mides d'imatge en que treballarem, la implementació paral·lela és de l'ordre de 10 vegades més ràpida.

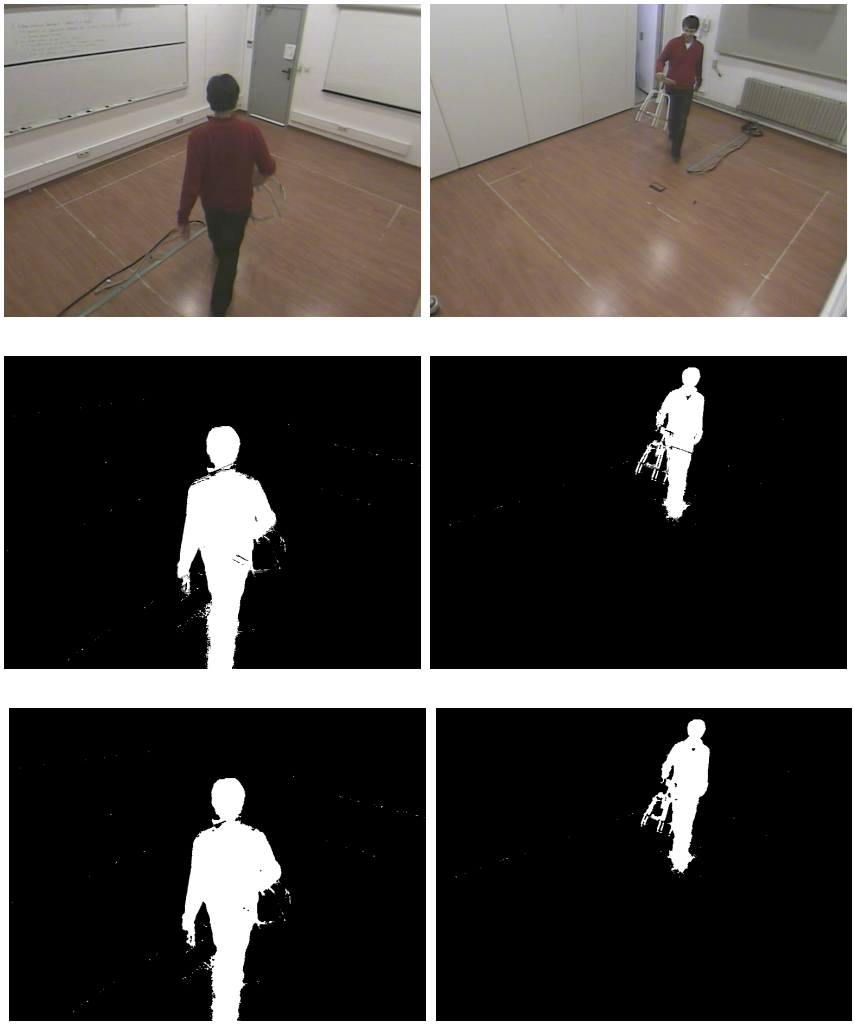

Figura 6.4: Segmentació de dues imatges. A la fila del mig versió sense tancament, a la de sota amb tancament d'ordre 1

La figura 6.4 mostra la segmentació de dues imatges de 752 columnes per 582 files. El tancament d'ordre 1 aconsegueix reduir talls en la segmentació de primer pla.

## 6.3 Reconstrucció d'escenes tridimensionals

L'objectiu d'aquest projecte és aconseguir una reconstrucció de superfície d'una escena capturada per N càmeres a temps real amb *GPU*. Aquest apartat presenta el nucli del problema: com paral·lelitzar l'algorisme per tal d'aconseguir processar uns 10 frames per segon.

En primer lloc es contextualitza la situació amb el rendiment del la implementació seqüencial, així com els objectius plantejats. Seguidament es presenten els resultats obtinguts en les diverses configuracions provades. En primer lloc es presentarà el plantejament inicial, pròxim a la implementació seqüencial. Posteriorment es presenten les diverses opcions explorades amb una justificació de les decisions preses. Finalment s'analitza el resultat definitiu.

### 6.3.1 Context i objectius

L'objectiu del projecte consisteix en aconseguir una reconstrucció de superfície en temps real. Cal concretar el marc d'operació i la situació de partida.

Es busca aconseguir una demostració en temps real en una sala equipada amb diverses càmeres. Es considera temps real processar uns 10 *frames* per segon.

Durant el projecte s'ha treballat amb diferents jocs d'imatges, tant pel que fa a la mida de les imatges, com al nombre de càmeres utilitzades en la reconstrucció. En aquest apartat, es treballa amb una configuració fixa, propera a la finalment implementada a temps real:

- Es treballa amb 7 càmeres. Es considera que és el número mínim per tal d'aconseguir una reconstrucció de superfície amb una certa qualitat
- Les càmeres proporcionen unes imatges de 752 columnes per 582 files entrellaçades. En la demostració les imatges són delmades a 376 columnes per 288 files
- Les imatges utilitzades són siluetes binàries ja segmentades prèviament
- El volum en el qual es farà la reconstrucció es limita a un cub de 2 metres d'aresta

El mètode extreu punts de superfície per cada joc d'imatges. Un primer pas és determinar el nombre de punts necessaris per tal d'aconseguir una reconstrucció

prou densa. La figura 6.5 mostra diverses reconstruccions de superfície amb un nombre creixent de punts. La imatge inferior esquerra presenta 50.000 punts de superfície, nombre suficient per aconseguir una bona representació.

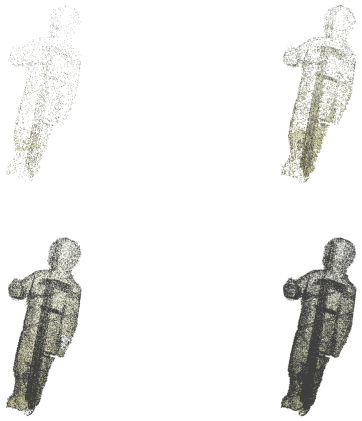

Figura 6.5: Reconstruccions de superfície amb nombre de punts creixents, de dalt a baix i d'esquerra a dreta: 3000, 10000, 50000 i 100000

La implementació seqüencial del mètode permet processar de 4 a 5 reconstruccions per segon amb 50000 punts de superfície, sense tenir en compte el post-processament. Tenint en compte el post-processament, el nombre baixa a 2 reconstruccions per segon.

En la implementació en paral·lel, no realitzarem les mateixes operacions que en el cas seqüencial, però caldrà tenir present aquest rendiment per comparar-lo amb les diferents versions desenvolupades amb OpenCL.

### 6.3.2 Plantejament inicial

La primera aproximació al problema va consistir en realitzar el càlcul de les agrupacions de punts veïns, tal com s'observa a la figura 6.6. Cada *work-item* s'encarrega d'un punt de superfície: busca un punt, els seus veïns i calcula l'orientació. El comportament d'aquesta primera prova era més lent que l'execució en *CPU*. Aquest comportament era degut als següents factors:

- Baixa probabilitat de trobar punts de superfície: la cerca aleatòria per tot l'espai ocasiona una probabilitat de trobar punts de superfície baixa.
- Per cada punt de superfície s'han de fer múltiples projeccions a les diverses imatges per determinar si la projecció forma part del primer pla i si algun

píxel veí forma part del fons. Aquest múltiples accessos a memòria, si la targeta té prou capacitat de càlcul, ocasionen que l'execució estigui limitada per l'ample de banda de la targeta gràfica.

- Una funció no acaba l'execució fins que tots els *work-item* d'execució no han acabat. Alguns *work-item* necessiten molts més intents que altres per trobar els punts de superfície, provocant que molts *work-item* s'esperin sense realitzar cap treball.

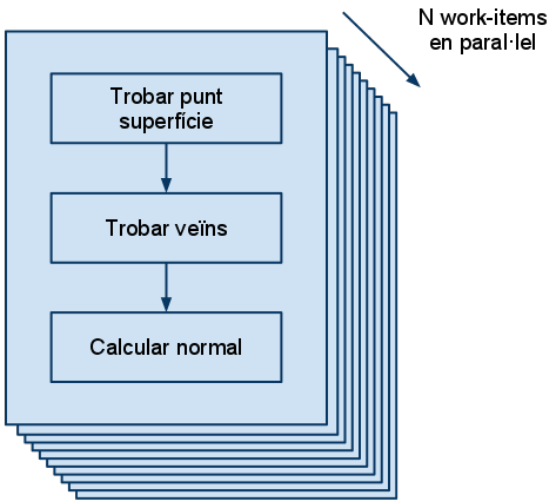


Figura 6.6: Plantejament inicial per l'execució paral·lela de l'algorisme

Un seguiment al nombre d'intents realitzats per cada fil mostra que amb uns pocs intents molts *work-item* ja han trobat punts de superfície mentre que altres *work-item* han de realitzar milers de cerques fins trobar el seu primer punt. També s'observa que un cop trobat un punt de superfície, per trobar els veïns calen un nombre d'intents molt més petit ja que la zona de cerca està molt més limitada i la probabilitat de trobar un punt de superfície és alta.

La figura 6.7 mostra el nombre de punts trobats en funció del nombre màxim d'intents permesos per cada *work-item* per un total de 100.000 *work-items*. Aproximadament amb 400 intents per cada *work-item* s'aconsegueix que la meitat dels *work-items* trobin un punt de superfície. El problema és que per trobar aquest punts de superfície el temps d'execució és de l'ordre dels 1,2 segons, molt superior a l'execució seqüencial.

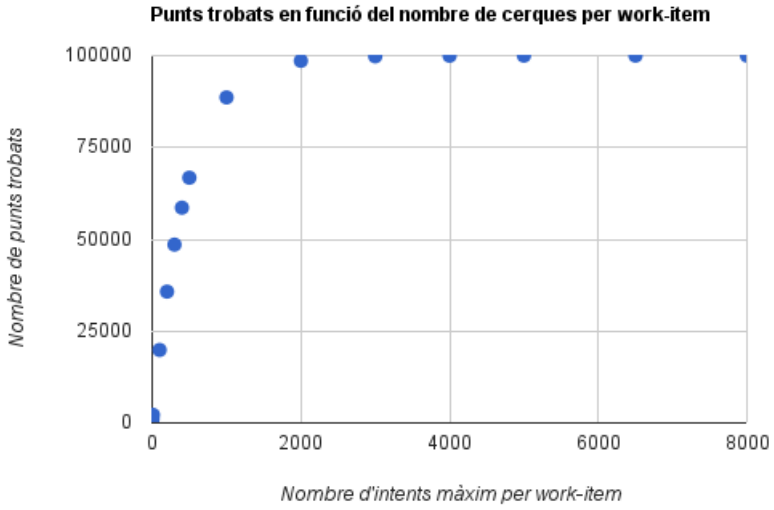


Figura 6.7: Durant la cerca aleatòria de punts, alguns *work-item* necessiten molts intents per trobar un punt de superfície, provocant que altres *work-items* s'hagin d'esperar

### 6.3.3 Alternatives explorades

En un primer intent de millora es va intentar utilitzar diferents configuracions per els *work-items*, les diferents proves amb *work-items* organitzats en diferents dimensions i mides no aporten millores.

Per tal de limitar l'espera fins que tots els *work-item* d'execució trobin un punt de superfície es fixa el nombre màxim de cerques a realitzar per cada fil. Cada fil té un nombre d'intents per trobar un punt de superfície, després d'aquests intents si no l'ha trobat desisteix. Aquesta limitació permet aconseguir un rendiment molt més alt però alhora presenta limitacions:

- Nombre de punts trobats aleatori: El mètode no permet fixar el nombre de punts a trobar. A cada frame el nombre de punts de la reconstrucció serà diferent.
- Control *CPU*: un cop buscats els punts, s'hauran de comprovar quants punts s'han trobat i reorganitzar-los per tal d'agrupar-los per càlculs posteriors. Aquest control permet repetir la cerca de punts si el nombre de punts trobat no és suficient.

Una altra línea de millora consisteix en realitzar la cerca dinàmica, aprofitant

el coneixement del *frame* anterior. Si la cerca dinàmica assolís un bon rendiment podríem basar-nos en ella per fer la majoria de reconstruccions, refrescant l'escena periòdicament amb una cerca exhaustiva.

Els beneficis de la cerca dinàmica consisteixen en poder limitar la zona de cerca al considerar que el moviment d'un *frame* al següent és limitat. Per tant, és possible limitar la cerca de punts a un espai proper a la superfície anterior. Amb aquesta configuració, els punts d'un *frame* a l'altre es busquen a partir de punts de superfície del *frame* anterior, podent aprofitar també la informació de la seva orientació.

La cerca dinàmica, si bé introdueix una millora respecte a la cerca aleatòria per tot l'espai, no permet un rendiment suficient per treballar amb temps real. Un altre problema tot i la millora de rendiment, és la dificultat d'agafar els punts llavor per realitzar l'expansió. Agafant-los aleatòriament entre els punts de la reconstrucció anterior provoca que els punts s'acumulin progressivament en zones on la probabilitat de superfície és major.

Aquest procés d'acumulació es fa visible ràpidament ja que, en reconstruir una persona, els punts de superfície s'acumulen progressivament a les cames, i es perd la reconstrucció en la zona del cap. Aquest comportament s'atribueix a què la probabilitat de trobar un punt de superfície és major al tronc i a la major mobilitat del cap i dels braços.

Analitzant les diferents alternatives i el rendiment insuficient que ofereixen, es determina que per tal de millorar el rendiment de la implementació en *GPU* és necessari reduir el nombre d'accessos a memòria. L'algorisme original assoleix un bon rendiment en *CPU*, les característiques de les *GPU* però, fan necessari replantejar l'estratègia de cerca.

La figura 6.8 mostra el model d'execució d'una *GPU* en comparació amb el model de *CPU*. L'execució en *CPU* redueix la latència amb l'ús de memòries *cache*. En el cas de les *GPUs* s'aconsegueix amagar la latència executant altres *work-item* d'execució mentre no estan disponibles les dades.

En l'algorisme original es realitzen múltiples accessos a memòria per cada punt candidat a esdevenir punt de superfície. Per tal de millorar el rendiment en *GPU* serà necessari reduir el nombre d'accessos a memòria tant com sigui possible.

Per tal de limitar l'espai de cerca reduint el nombre d'accessos a memòria, es pot realitzar un *Visual Hull* voxelitzat de l'escena. En el càlcul del *Visual Hull* voxelitzat es mostreja regularment l'espai cercant zones de l'escena ocupades per

algun volum. Al buscar punts ocupats en comptes de punts de superfície el nombre de projeccions a les diverses càmeres és molt menor, reduint el nombre d'accessos a memòria i augmentant el rendiment de l'algorisme.

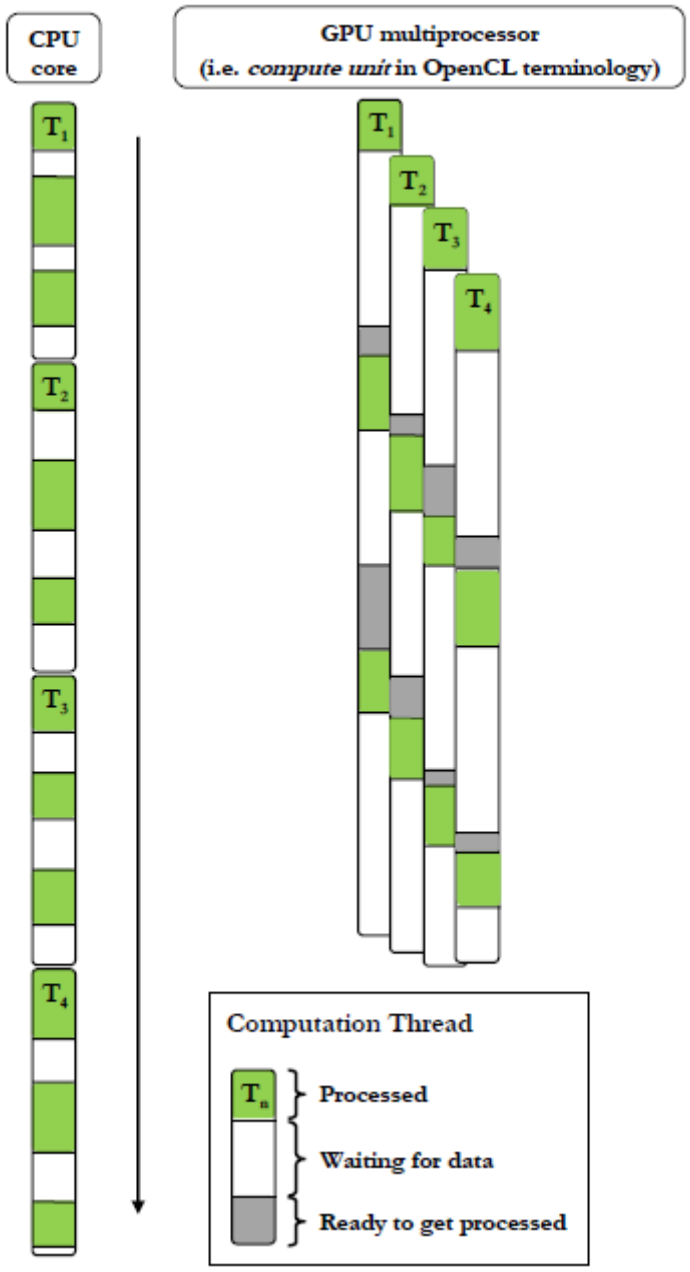


Figura 6.8: Per execucions amb molts càlculs en paral·lel, les *GPUs* assoleixen millors rendiments amagant la latència amb processament d'altres dades, en lloc de reduir la latència amb una cache com fan les *CPUs*

El resultat del *Visual Hull* voxelitzat és un conjunt de punts tridimensionals

ordenats amb informació de si el volum es troba ocupat o no. Aquest conjunt de punts s'acostumen a anomenar vòxels, en aquest cas s'utilitzen com a suport per a cercar punts de superfície. Per tal de no confondre termes, es referirà al resultat de la voxelització com a *bins* en lloc de vòxels. Aquesta nova notació fa palès que la voxelització és un pas intermedi de l'algorisme (les 'caixes' o 'regions' on es buscaran els punts de superfície) i no un resultat.

Un cop es disposa dels *bins* i al obtenir-los d'una voxelització regular, es poden recórrer les posicions adjacents per detectar si estan ocupades, podent elegir només els *bins* de superfície: *bins* que estan ocupats i que limiten amb *bins* buits.

Els bins de superfície possibiliten reduir l'espai de cerca de punts de superfície a un radi d'una mida de l'ordre de la separació entre bins.

### 6.3.4 Solució proposada

La solució proposada presentada a 5.3.2 aconsegueix reduir el nombre d'accessos a memòria, que és el factor limitant de l'algorisme. La voxelització prèvia a la cerca de punts de superfície permet acotar l'espai de cerca de punts de superfície, aconseguint una probabilitat més alta de cerca de punts. En la implementació realitzada, la voxelització realitzada és amb vòxels d'amplada 4 centímetres en cada dimensió.

La figura 6.9 mostra el nombre de punts trobats en una execució amb 100.000 *work-items* buscant en paral·lel al variar el nombre d'intents màxims permès a cada *work-item*. L'eficiència de cerca de punts augmenta considerablement respecte a buscar en tot el volum: amb la voxelització prèvia, amb 10 intents per *work-item* es troben més de la meitat de punts cercats, mentre que cercant en tot el cub de 2 metres d'aresta es necessiten 400 cerques per *work-item* per assolir aquesta xifra.

#### Operacions detallades

A la figura 6.10 es pot observar les operacions realitzades en la reconstrucció de superfície. Es tracta de les tasques a realitzar cada *frame*, prèviament és necessari crear les funcions i els buffers necessaris. Les operacions es llisten a l'esquerra del diagrama i es detallen a continuació.

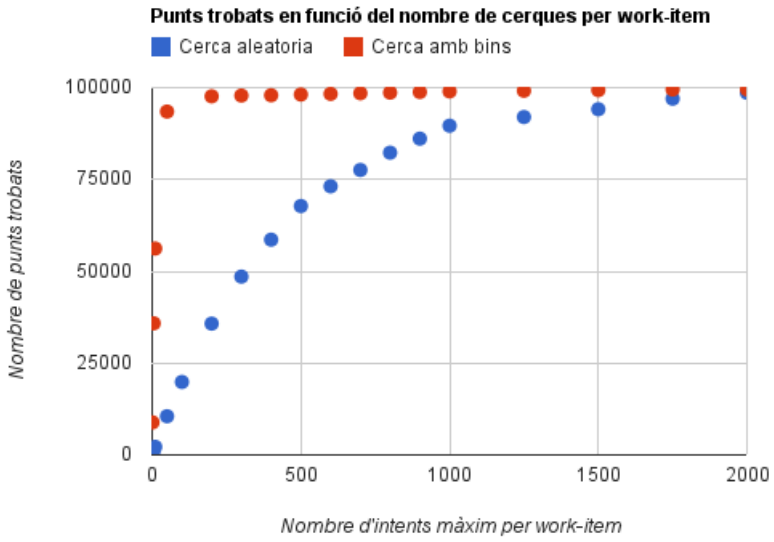


Figura 6.9: L'eficiència de cerca de punts augmenta en cas d'utilitzar una voxelització prèvia

### 1 Transferència segmentació de primer pla

La segmentació de primer pla és la base de la reconstrucció 3D. El primer pas és disposar de les dades a *GPU*. Cada imatge té unes dimensions *height* i *width*. Per guardar la informació de la segmentació de primer pla bastaria amb un bit per a cada píxel. Per consistència amb la implementació en *CPU* i per no estar especificat el tipus *bool* a la *API* del *host* s'emmagatzema en 1 byte. La transferència de dades té una mida total de: Mida buffer segmentació = $Num\_cameres * height * width * 8$ $[bits]$

### 2 Kernel Visual Hull Voxelitzat

Cada *work-item* processa un vòxel de l'escena. Les dimensions de l'espai de cerca són fixades, a partir de l'identificador i la posició del màxim i del mínim de l'escena, es calcula la posició 3D del vòxel.

El vòxel es projecta a les diverses càmeres. Teòricament, un vòxel es considera ocupat si totes les càmeres el detecten com a ocupat. En la implementació es permet un error per possibles errors en la segmentació de primer pla. Com en el cas de la segmentació amb un bit es podria codificar la informació d'un vòxel, però es realitza amb un byte.

### 3 Kernel detectar bins de superfcie

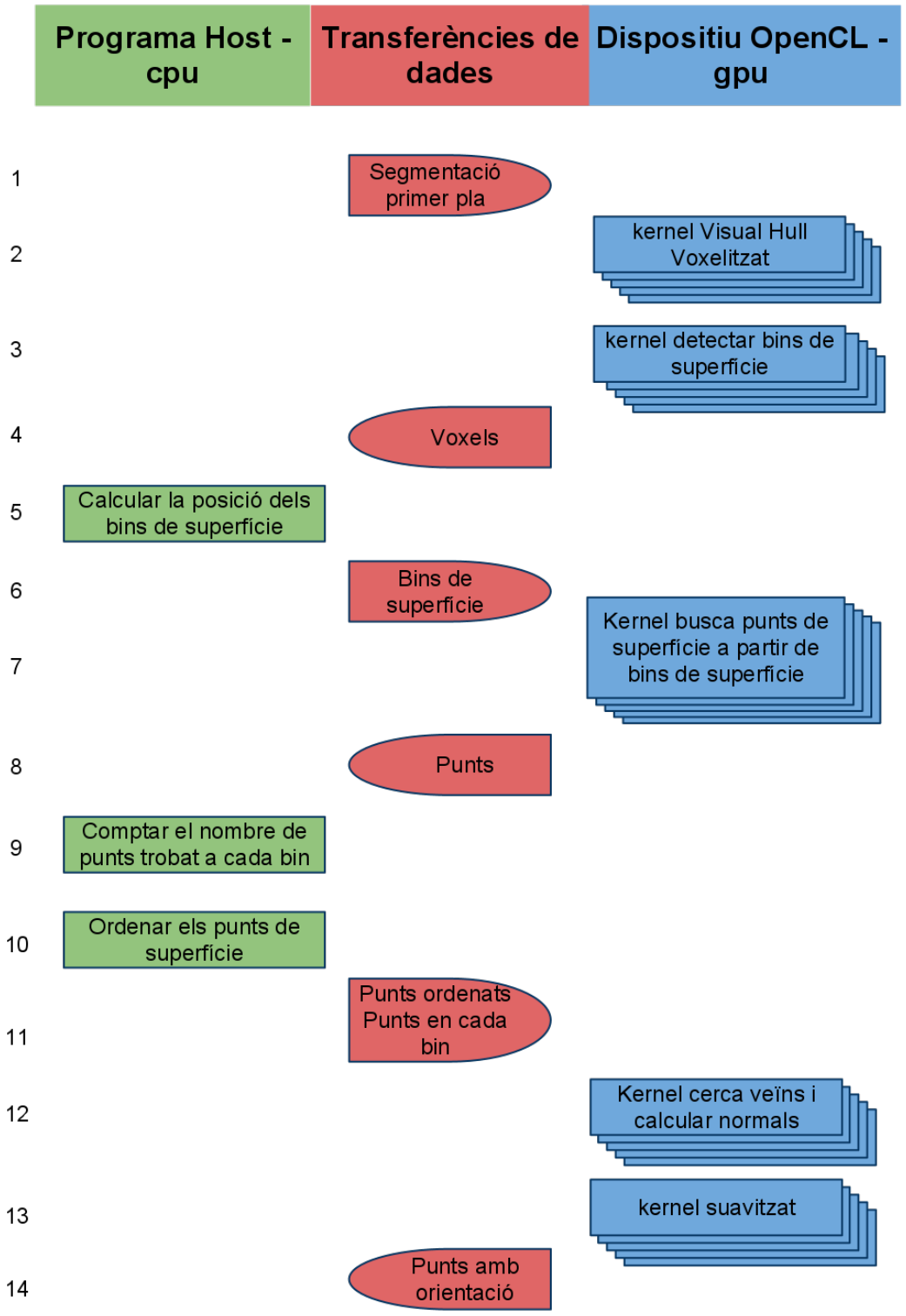


Figura 6.10: Diagrama d'operacions en la reconstrucció 3D de superfícies en *GPU*

L'ordenació de la voxelització permet consultar els vòxels veïns per comprovar si es troben ocupats. D'aquesta forma es poden detectar els vòxels de superfície. Aquesta operació permet descartar vòxels que es troben a la zona interior d'un objecte, on no es troba cap punt de superfície. La comprovació de vòxels ocupats es realitza als 6 vòxels adjacents.

**4 Transferència dels vòxels**

Es transfereixen els vòxels a *CPU* per tal de cercar els vòxels de superfície. La mida dels vòxels és:
Mida buffer vòxels = $n\_voxels\_dim\_x * n\_voxels\_dim\_y * n\_voxels\_dim\_z * 8$ $[bits]$

**5 Calcular la posició dels bins de superfície**

A partir dels vòxels marcats com a ocupats, es recupera la posició central del vòxel a la que anomenem bin, ja que seran els punts base per realitzar la cerca de punts de superfície. Es tracta d'una operació que es podria paral·lelitzar i ser realitzada en *GPU*. Caldria tenir en compte la sincronització dels *work-items* i només es podria utilitzar un *work-group*. La realització amb *CPU*, tot i les transferències de memòria, es prou ràpida com per no requerir la paral·lelització.

**6 Transferència dels bins de superfície**

Un cop calculats els bins en *CPU*, es transfereixen a la *GPU*. A cada *frame* el nombre de bins trobats serà diferent. Cada bin es codifica amb un $cl\_float\_4$ (un cl_float_4 té una mida de 4*32 bits. La transferència total és de: Mida buffer bins = $nombrebinstrobats * 4 * 32$ $[bits]$

**7 Kernel busca punts de superfície**

A partir dels bins de superfície es busquen punts de superfície en una esfera de radi la separació entre bins. Aquesta separació es prou petita com per tenir una bona probabilitat de trobar punts de superfície així com prou gran per evitar possibles errors en el mostreig regular en bins Es tria un radi que sigui prou reduït per tenir una bona eficiència i que sigui robusta a errors en la voxelització. Amb el radi elegit, si dos bins adjacents estan ocupats, tenen una superposició parcial que evita àrees de superfície sense punts.

El nombre de *work-items* de cerca es fixa abans de l'execució. Els *work-items* es reparteixen automàticament entre els diferents bins ja que cada *frame* el nombre de bins serà diferent. Cada *work-item* genera un punt aleatori dins l'esfera. Quan un *work-item* troba un punt de superfície, guarda el número de bin al qual pertany.

**8 Transferència de punts**

Els punts de superfície trobats amb la *GPU* es transfereixen novament a *CPU* per trobar quants punts es troben en cada bin i evitar espais buits en la cerca de veïns als kernels posteriors. Cada punt es codifica amb un $cl\_float\_4$ (un cl_float_4 té una mida de 4*32 bits. La transferència total és de: Mida buffer punts = $nombrepuntsbuscats * 4 * 32$ [$bits$]

**9 Recompte de punts a cada bin**

Per tal de buscar els veïns s'utilitzarà el coneixement de l'estructura en bins. Per tal de realitzar la cerca cal determinar el nombre de punts en cada bin. El càlcul en *CPU* torna a ser més còmode per aquest fet.

**10 Ordenar els punts de superfície**

Els punts de superfície trobats es concentren en un vector per tal de reduir la cerca de veïns realitzada al *kernel*.

**11 Transferència de punts ordenats**

Es transfereixen els punts ordenats anteriorment així com el vector que conté el nombre de punts a cada bin. El nombre de punts en cada bin s'emmagatzema en un vector de cl_uint que ocupa 16 bits cada element. El vector de bins indica els índexs màxims i mínims del vector de punts per cada bin. Mida buffer punts = $nombrepuntstrobats * 4 * 32$ [$bits$] Mida buffer bins = ($nombredebinstrobats$ + 1) $* 16$ [$bits$]

**12 Kernel càlcul de normals**

L'organització en *bins* permet buscar dins del*bin* tots els punts situats a una certa distància. Per cada punt de superfície trobat, es busca dins del bin tots els punts que es troben a una distància menor a un llindar. Un cop trobats tots els veïns es calcula l'orientació del punt.

Amb el radi de cerca de punts utilitzat, hi ha punts propers que no es poden utilitzar com a veïns ja que provenen d'un bin veí. Aquesta implementació augmenta el soroll en el càlcul de les normals.

**13 Kernel suavitzat**

Un cop calculada l'orientació de cada punt es pot realitzar el suavitzat de superfície. Es tornen a buscar tots els veïns a una distància menor a un llindar

que pot ser diferent al cas anterior. Amb les normals calculades anteriorment es realitza un desplaçament dels punts en direcció a la normal en funció de la posició dels seus veïns.

**14 Punts amb orientació**

Es transfereixen els punts amb les orientacions cap a *CPU* com a pas final. Les normals es codifiquen també amb vectors de $cl\_float\_4$. Mida buffer punts = mida buffer normals = nombre punts trobats$*4*32$ $[bits]$

### 6.3.5 Anàlisi de temps

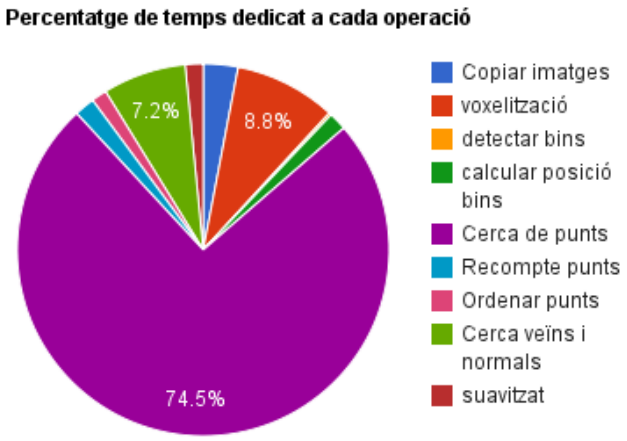


Figura 6.11: Gràfic diverses operacions realitzades en la reconstrucció de superfície

El llistat d'operacions és molt extens, tot i això la major part del temps es destina a la cerca de punts de superfície com mostra la figura 6.11. La voxelització prèvia, si bé introdueix un cost computacional elevat, és assumible en comparació a la resta de l'algorisme. Es raonable pensar en ampliar l'espai de cerca, ja que presumiblement augmentarà el temps destinat a la voxelització, i no així el temps de cerca de punts.

| Operació | Temps mitjà (ms) |
|---|---|
| Transferència siluetes | 3,456 |
| Voxelització | 10,133 |
| Detectar bins | 0,255 |
| Calcular posició bins | 1,788 |
| Cerca de punts de superfície | 85,367 |
| Recompte punts | 2,033 |
| Ordenar punts | 1,535 |
| Cerca veïns i normals | 8,301 |
| Suavitzat de superfície | 1,785 |
| **Total** | **114,653** |

Figura 6.12: Temps requerit per cada operació en la reconstrucció de superfície

Cal destacar el compromís en la voxelització i la cerca de punts de superfície. Treballant amb vòxels més petits, la cerca de punts és més eficient però el temps de voxelitzar l'escena augmenta.

Les operacions realitzades en *CPU* no representen un cost computacional elevat respecte a les operacions realitzades en *GPU* com es pot observar a 6.12. Es tracta d'operacions de reorganització de les dades, en les quals la *CPU* és molt eficient. Els temps de transmetre les dades entre la *CPU* i la *GPU* es troben inclosos en les operacions en *CPU*. La transferència inicial de les siluetes sí que es detalla pel seu major pes.

### 6.3.6 Qualitat reconstrucció

Fins el moment, s'ha presentat la comparació de temps entre la implementació seqüencial i la paral·lela. En aquest punt s'exposa que amb l'execució paral·lela s'assoleix un rendiment suficient per la seva visualització.

En primer lloc es presenten els resultats de l'execució seqüencial per tenir una referència als resultats obtinguts.

#### Reconstrucció de superfície: execució seqüencial

La figura 6.13 mostra la reconstrucció de superfície obtinguda amb el mètode seqüencial. La reconstrucció de l'esquerra mostra la reconstrucció sense l'etapa de post-processament, per tant les normals són estimades a partir d'uns pocs veïns i la superfície no ha estat suavitzada. A la dreta s'observa la mateixa reconstrucció amb l'etapa de post-processament realitzada.

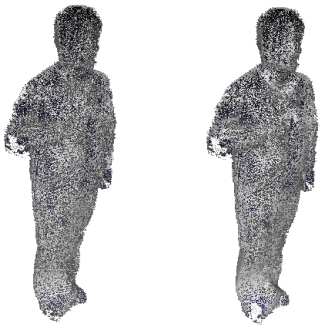

Figura 6.13: Reconstrucció de superfície obtinguda amb el mètode seqüencial. La reconstrucció de l'esquerra sense suavitzat, la de la dreta amb suavitzat

La comparació entre les dues imatges permet apreciar els avantatges de l'etapa de post-processament. En les figures es representen els punts juntament amb la seva orientació. Els punts visualitzats en la direcció cap a on estan orientats s'observen en color blanc mentre que si l'orientació és oposada s'observen en gris. A la reconstrucció post-processada un major percentatge de punts s'observen en blanc.

Els punts que a la reconstrucció post-processada s'observen en gris provenen de punts situats a la cara oposada de la reconstrucció. Això es deu a que la reconstrucció no es prou densa i en algunes zones s'observen els punts de l'altre

cara. Per tal de solucionar aquest problema, si no és possible augmentar el nombre de punts de superfície, caldrà augmentar la superfície de cada punt dibuixat.

La reconstrucció de la dreta també permet observar l'efecte del suavitzat. El soroll present al mètode de cerca aleatori s'ha reduït. En la silueta de la reconstrucció i a les cames s'observa que els punts es troben en una mateixa superfície.

### Reconstrucció de superfície: execució paral·lela

En la execució paral·lela es calculen les normals conjuntament, la separació entre cerca de punts i post-processament no té sentit en aquest cas. És per això que es compararà l'execució paral·lela final amb l'execució seqüencial amb i sense l'etapa de post-processament.

La figura 6.14 mostra la comparació entre l'execució seqüencial sense post-processament i la versió paral·lelitzada. Es pot observar com la versió paral·lelitzada sembla presentar un nombre major de punts, aquesta percepció és deguda a que el mètode seqüencial assigna la mateixa normal al punt i als veïns utilitzats en el seu càlcul. Un cop realitzat l'acoloriment dels punts aquesta percepció desapareix.

Figura 6.14: Comparació reconstrucció de superfície obtinguda amb els mètodes seqüencial i paral·lel. La reconstrucció de l'esquerra correspon al seqüencial, a la dreta el paral·lel

La implementació paral·lela, si bé presenta una bona distribució de punts en general, presenta zones amb densitat menor de punts. Aquest comportament es deu a la voxelització realitzada. Si hi ha algun error en la detecció dels vòxels ocupats, la superposició parcial en la cerca de punts permet evitar zones sense punts de superfície, però poden quedar zones amb menor densitat de punts.

Figura 6.15: Efecte del suavitzat de superfície. La reconstrucció de l'esquerra és l'obtinguda amb el mètode paral·lel. La versió del mig consisteix en la mateixa reconstrucció suavitzada utilitzant tots els punts de superfície. A la dreta s'observa la versió seqüencial suavitzada

A la figura 6.14 es pot observar que el suavitzat realitzat a la reconstrucció, si bé no tan eficaç com el suavitzat seqüencial permeten aconseguir una superfície menys sorollosa.

Els resultats del suavitzat de superfície poden ser observats amb més precisió en la figura 6.15. Es compara el suavitzat de superfície obtingut amb el mètode paral·lel (a l'esquerra) amb el mètode seqüencial.

Cal recordar que el suavitzat en el mètode paral·lel es realitza tenint en compte únicament els punts veïns dins del mateix bin. Per tant el suavitzat de superfície paral·lelitzat obté pitjors resultats com és d'esperar.

A la mateixa figura 6.14, al centre, es mostra la reconstrucció que s'obtindria amb el mètode paral·lelitzat si cada punt busqués en tots els punts els possibles veïns. Es pot observar que el suavitzat de superfície obtingut millora. Es tracta d'una comprovació que el suavitzat es realitzat correctament, no pot ser portada a terme en la reconstrucció a temps real ja que el cost computacional és molt elevat.

### Acolorir els punts de superfície

La qualitat del resultat d'acolorir els punts de superfície ve determinada per la validesa de l'orientació de cada punt. Si l'orientació es troba ben estimada, l'acoloriment es realitzarà amb les càmeres més ben alineades amb el punt, obtenint un millor resultat.

Figura 6.16: Acoloriment dels punts de superfície a partir d'una reconstrucció seqüencial. A l'esquerra versió sense suavitzat de superfície, a la dreta reconstrucció suavitzada

Figura 6.17: Acoloriment dels punts de superfície a partir d'una reconstrucció paral·lela. A l'esquerra versió amb suavitzat de superfície en temps real, a la dreta reconstrucció suavitzada tenint en compte tots els punts

La figura 6.16 mostra l'acoloriment de les reconstruccions de superfície de la figura 6.13. Els punts en blau corresponen a forats a la representació, la reconstrucció no es prou densa amb la mida de punts utilitzats.

La figura 6.17 mostra l'acoloriment de les reconstruccions presentades en 6.15. En ambdós casos es fa patent el soroll present en l'orientació de les normals. Alguns punts troben pocs veïns dins del bin, la qual cosa ocasiona un càlcul de normals poc precís. Al acolorir els punts, aquest soroll es fa palès, obtenint punts acolorits incorrectament.

En la imatge presentada, el soroll en l'orientació es reflexa amb una baixa qualitat de la representació. Al observar la reconstrucció en temps real, es visualitzen moltes imatges amb aquest tipus de soroll, el que es tradueix en una

percepció global de soroll en els punts trobats. És per això que al següent capítol es proposa un acoloriment alternatiu dels punts de superfície.

La comparació amb l'execució seqüencial permet veure que l'orientació calculada a partir d'uns pocs veïns molt propers al punt obté millors resultats que buscant els punts veïns en el bin de l'execució paral·lela.

### 6.3.7 Resultats obtinguts variant l'espai de cerca

L'eficiència en la cerca de punts de superfície depèn de la mida de l'espai de cerca on es realitza la reconstrucció. El desenvolupament de l'algorisme s'ha realitzat en una àrea de cerca de $8m^3$.

Es va considerar aquesta mida de cerca per ser un volum suficient per contindre una persona i poder observar els seus moviments en un espai limitat. Alhora, s'estimava que amb aquesta mida es podria aconseguir una reconstrucció de superfície a temps real.

En aquest punt es realitza un experiment variant les dimensions de cerca, comparant el rendiment amb l'obtingut amb el mètode seqüencial. El nombre de punts per a cada reconstrucció és de 100.000 punts, s'analitzen els resultats obtinguts amb les mateixes imatges d'entrada, variant la mida de l'espai de cerca.

El mètode seqüencial s'analitza tant en la versió d'inicialització com en la dinàmica, la qual s'ajuda d'una reconstrucció anterior per trobar els punts de superfície llavors. Per la realització d'aquest experiment s'ha considerat un radi d'expansió constant en la cerca dinàmica. Aquest radi constant possibilita una bona qualitat en la reconstrucció, alhora d'un millor rendiment respecte a la inicialització.

Figura 6.18: Resultat obtingut amb el mètode seqüencial dinàmic amb cerca a tota la sala. A l'esquerra sense post-processament, a la dreta amb l'estimació de normals i el suavitzat posterior

El resultat obtingut en la cerca per tota la sala s'observa a la figura 6.18. L'etapa de post-processament permet assolir una gran qualitat però amb un temps

de processat superior als dos segons per a cada imatge.

En el desenvolupament de l'algorisme paral·lel es van fixar els paràmetres que aconseguien una millor eficiència de cerca del sistema complet per l'espai de cerca elegit. Augmentant el nombre de vòxels en cada dimensió s'aconseguiria reduir l'espai de cerca de punts, però a costa d'augmentar el cost de la voxelització. Per contra, reduir el nombre de vòxels augmentaria l'espai de cerca de punts.

Figura 6.19: Resultat obtingut amb el mètode paral·lel amb cerca a tota la sala. A l'esquerra amb el nombre de vòxels constant, a l'esquerra amb la mida dels vòxels constant

Al augmentar l'espai de cerca existeixen dues possibilitats amb l'algorisme paral·lel. Una primera consisteix en mantenir el nombre de vòxels total constant. D'aquesta forma s'aconsegueix mantindre el temps de la voxelització inicial constant, però a costa de disminuir l'eficiència de cerca de punts de superfície. A la figura 6.19 a l'esquerra s'observa el resultat obtingut amb aquesta configuració.

La segona possibilitat consisteix en mantindre la mida dels vòxels constant, aconseguint mantenir l'eficiència de cerca de punts de superfície, augmentant el cost de la voxelització inicial. A la figura 6.19 a la dreta s'observa el resultat d'aquesta configuració.

Com es pot observar la qualitat de la reconstrucció és similar en les dues aproximacions al problema. El post-processament paral·lel permet reduir el soroll inherent al mètode de cerca de punts amb el temps limitat del que disposa degut al seu enfocament a temps real.

La comparació dels temps d'execució es pot observar en el gràfic 6.20. El mètode paral·lel aconsegueix millorar el rendiment del mètode seqüencial entre

1.5 i 2 vegades. Al augmentar el nombre de punts de l'experiment respecte a l'anterior la millora del mètode paral·lel ha disminuït. Aquest fet és degut a què el mètode seqüencial millora l'eficiència a través d'expansions de punts ja trobats.

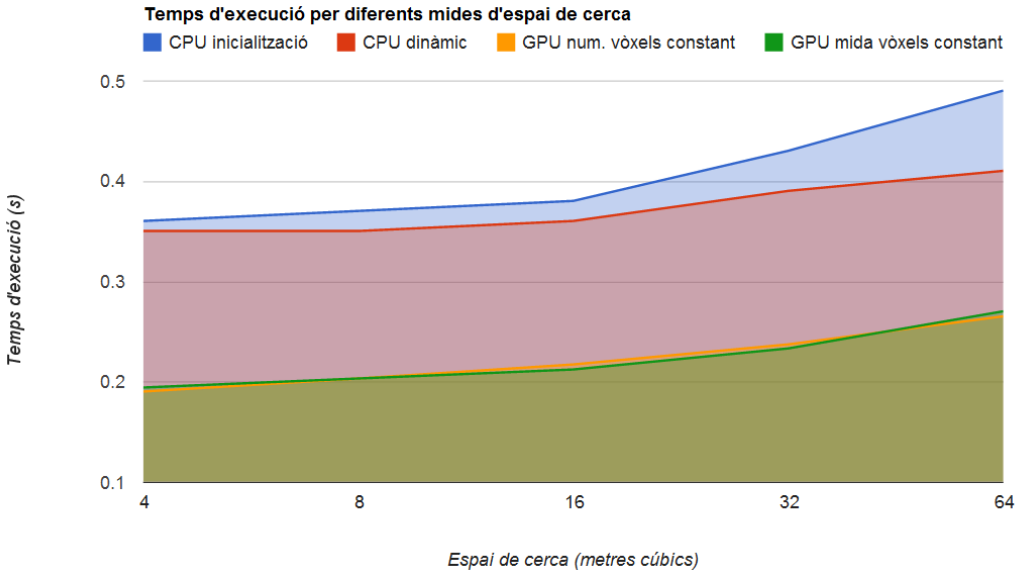


Figura 6.20: Resultat obtingut amb el mètode seqüencial dinàmic amb cerca a tota la sala. A l'esquerra sense post-processament, a la dreta amb l'estimació de normals i el suavitzat posterior

També cal recordar que els temps del mètode paral·lel inclouen el post-processament, mentre que el seqüencial no l'inclouen, ja que augmenta el temps d'execució un ordre de magnitud.

Els avantatges del mètode seqüencial dinàmic augmenten amb dimensions de la sala majors, tot i així el rendiment assolit amb el mètode paral·lel és millor.

El rendiment obtingut per les dues versions paral·leles és molt similar. Aquest fet permet corroborar l'eficiència del mètode desenvolupat amb independència dels paràmetres elegits. Per tal d'elegir entre una i altra alternativa, mantenir el nombre de vòxels constant pot millorar lleugerament la qualitat de la reconstrucció ja que obtindrem bins majors, limitant els problemes en les zones d'unió de bins. En canvi, augmentar el nombre de vòxels permet aconseguir un mètode més robust, ja que es mostregen major nombre de punts, aconseguint millor precisió en volums de poca amplària.

### 6.3.8 Comparació resultats en temps real

Per tal de realitzar una comparació justa entre l'implementació seqüencial i la paral·lela és necessari observar els resultats obtinguts per un mateix temps d'execució.

Amb aquest objectiu es comparen els resultats obtinguts entre les diverses execucions per un mateix joc d'imatges. La zona de cerca per les diverses proves és tota la *smartroom* amb una limitació temporal que permeti treballar amb 8 reconstruccions per segon. Per tant, el temps d'execució es limita a 0,125 segons.

Amb aquest temps d'execució la implementació seqüencial no pot realitzar cap tipus de suavitzat de superfície. La versió d'inicialització es capaç de trobar uns 30.000 punts de superfície en aquest temps, mentre que la versió dinàmica n'assoleix 35.000.

Pel que fa a la implementació paral·lela (versió amb vòxels més grans) el nombre de punts de superfície trobats és de 53.000, podent realitzar el suavitzat de superfície presentat. A la figura 6.21 es poden observar els resultats obtinguts a partir de diversos jocs d'imatges.

Els resultats permeten observar que la densitat de punts obtinguda amb el mètode seqüencial no és suficient per obtenir una correcta visualització. Amb el mètode paral·lel la densitat de punts ha augmentat, tot i que encara existeixen zones on la superfície presenta obertures.

L'efecte del suavitzat de superfície es fa difícil de percebre en la implementació paral·lela. Tot i això cal destacar que el temps invertit en el suavitzat de superfície paral·lel és molt reduit (menor d'un 2% del temps d'execució de la reconstrucció completa). La divisió en vòxels possibilita realitzar un suavitzat de superfície d'alt rendiment però amb una qualitat limitada.

Figura 6.21: Comparació de reconstruccions en temps real. A l'esquerra resultat obtingut amb el mètode seqüencial d'inicialització, al centre amb el mètode seqüencial dinàmic i a la dreta amb el mètode paral·lel

### 6.3.9 Resultats obtinguts amb diferents jocs d'imatges

Fins el moment s'han presentat resultats amb jocs d'imatges procedents de la sala intel·ligent (veure capítol 7). En aquest punt es presenten els resultats obtinguts per altres jocs d'imatges.

#### Reconstrucció a partir de 18 càmeres

El joc d'imatges en aquest cas consisteix en 18 imatges de mida 640x480 píxels. Les proves realitzades s'han realitzat amb 100.000 punts de cerca i un espai de cerca limitat a $8m^3$. La figura 6.22 mostra els resultats obtinguts.

Els resultats obtinguts assoleixen una qualitat superior als experiments amb 7 càmeres com era d'esperar. Els talls observats en el braç són deguts a que el fons de les imatges presenta moltes zones amb fortes reflexions, impossibilitant una correcta segmentació de primer pla.

Figura 6.22: Resultats amb 18 càmeres. A l'esquerra mètode seqüencial, al centre seqüencial amb post-processament, a la dreta paral·lel

El rendiment obtingut per el mètode seqüencial és de 0.82 segons per reconstrucció en la inicialització i de 0.60 segons en la cerca dinàmica. En ambdós casos sense considerar el post-processament.

Pel que fa al mètode paral·lel, el temps mitjà de procés és de 0.32 segons per reconstrucció. Com en el cas de treballar amb 7 imatges, el rendiment del mètode paral·lel és superior. En aquest cas aconseguint una millora entre 2 i 3 vegades del mètode seqüencial.

### Reconstrucció a partir de 16 càmeres

En el punt anterior s'ha realitzat un experiment augmentant el nombre de càmeres. En aquest apartat es prova el rendiment al treballar amb imatges de major resolució.

El joc d'imatges consisteix en 16 imatges de mida 1624x1224 píxels. Les proves realitzades s'han realitzat amb 100.000 punts de cerca i un espai de cerca limitat a $8m^3$. La figura 6.23 mostra els resultats obtinguts.

Figura 6.23: Resultats amb 16 càmeres. A l'esquerra mètode seqüencial, al centre seqüencial amb post-processament, a la dreta paral·lel

El rendiment obtingut per el mètode seqüencial és de 1.55 segons per reconstrucció en la inicialització i de 1.31 segons en la cerca dinàmica. En ambdós casos sense considerar el post-processament.

Pel que fa al mètode paral·lel, el temps mitjà de procés és de 0.73 segons per reconstrucció. Les millores en el rendiment obtingudes són similars al treballar amb 7 o 16 imatges. Treballar amb càmeres de major resolució ocasiona una probabilitat de trobar punts de superfície menor. En la implementació paral·lela implica transmetre les imatges a la *GPU*, tot i l'augment de la mida de les imatges el rendiment global segueix millorant al mètode seqüencial.

En aquest capítol s'ha validat l'objectiu del projecte d'aconseguir una millora de rendiment amb la implementació paral·lela suficient per a l'execució en temps real, comprovant que aquesta implementació no suposa una pèrdua de qualitat de la reconstrucció.

# Capítol 7

# Integració i arquitectura d'un demostrador

## 7.1 Smart Room

Una *smart room* o sala intel·ligent és una sala equipada amb diversos sensors audio-visuals i servidors. Els sensors recullen informació sobre les persones que es troben al seu interior, proporcionant dades als servidors, els quals analitzen les dades i interactuen amb els usuaris. En una *smart room* es combinen tecnologies visuals i acústiques per tal de proporcionar diferents serveis als usuaris.

La *smart room* de la UPC està equipada amb 7 càmeres fixes més 2 càmeres mòbils. Disposa de diverses agrupacions de micròfons distribuïts per la sala. La

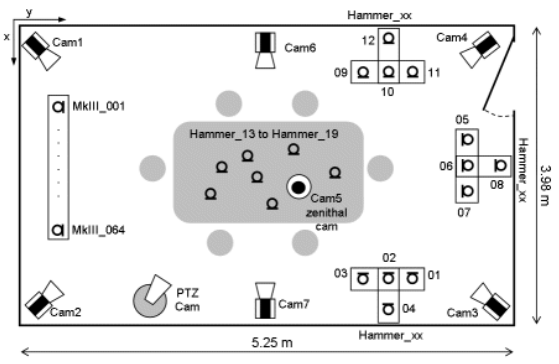


Figura 7.1: Planta de la *smart room*

figura 7.1 mostra la disposició de la sala.

Pel present projecte es pretén utilitzar les 7 càmeres fixes: una càmera zenital i 6 col·locades als extrems de la sala per tal d'aconseguir una reconstrucció en temps real. Les càmeres són del model JVC TK-C1481EG, proporcionant imatges de 752 columnes i 582 files entrellaçades.

La sala disposa de 5 servidors per processar les dades de les càmeres i generar els resultats per interactuar amb els usuaris. Els servidors estan equipats amb processadors Intel(R) Core(TM)2 Duo CPU E7400.

Les implementacions realitzades s'executaran en targeta gràfica, tenint una gran importància la *GPU* de cada màquina. Les característiques de les targetes gràfiques utilitzades es llisten a continuació.

La sèrie Quadro de NVIDIA està pensada per aplicacions de disseny, animació i vídeo. S'especialitza en aplicacions de visualització computacional complexes:

- Dispositiu: Quadro FX 3700
- Fabricant: NVIDIA Corporation
- Nuclis de processament: 112
- Velocitat de rellotge: 1250 MHz

La sèrie GeForce GTX de NVIDIA proporciona un elevat rendiment i prestacions per tot tipus d'aplicacions:

- Dispositiu: GeForce GTX 295
- Fabricant: NVIDIA Corporation
- Nuclis de processament: 240
- Velocitat de rellotge: 1242 MHz

Finalment, 3 servidors estan equipats amb GeForce 8400 GS, targetes poc potents en comparació a les altres dues però que permeten realitzar la segmentació de primer pla i la desdistorsió de les imatges.

- Dispositiu: GeForce 8400 GS

- Fabricant: NVIDIA Corporation
- Nuclis de processament: 8
- Velocitat de rellotge: 1125 MHz

## 7.2 SmartFlow

En aplicacions amb múltiples sensors i una elevada càrrega computacional és necessari processar les dades amb múltiples màquines. Per aquest propòsit és necessari disposar d'una sincronització entre els diferents dispositius.

La tecnologia elegida a la *smart room* per la sincronització entre les diverses màquines és *smartflow*. *Smartflow* possibilita la interacció de diverses màquines comunicant-se entre elles amb el protocol ssh. Aquestes crides són internes a *smartflow*. La configuració d'una aplicació amb *smartflow* es realitza amb dos fitxers de configuració:

- Configuració general: llista xml amb els servidors i els *paths* dels clients que es pretén utilitzar
- Mapa d'aplicació: Conté els clients utilitzats, les seves connexions, els paràmetres de configuració i el nom dels *flows*

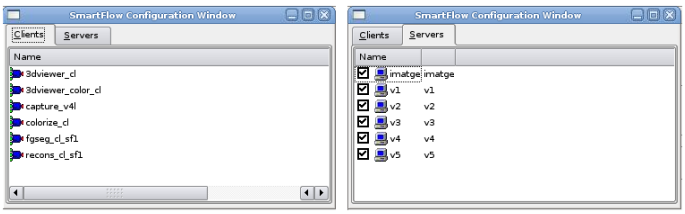


Figura 7.2: Interfície de selecció dels clients i servidors

Els dos fitxers poden ser actualitzats manualment o a través d'una interfície gràfica que proporciona *smartflow*. A la figura 7.2 es mostra la llista de clients i de servidors.

La programació dels diversos clients es realitza en fitxers C++ on s'especifica el format dels *flows* d'entrada i sortida, així com les operacions a realitzar en cada *frame* i el moment de rebre i enviar els *flows*.

En l'aplicació desenvolupada es tracten 7 càmeres simultàniament. Per il·lustrar el funcionament de *smartflow* es descriu el contingut a cada fitxer que fa possible capturar la informació de les diverses càmeres:

- Configuració general: s'especifica el nom pel qual serà reconegut el client (capture_v4l), el *path* del seu binari i descriu els *flows*: en aquest cas disposa de dos *flows* de sortida, un amb les imatges capturades i un altre que proporciona marques temporals

- Mapa d'aplicació: Cada client capture_v4l especifica el servidor on s'executa, la capturadora de vídeo utilitzada, la mida de les imatges proporcionades, el fitxer de calibració per cada càmera i el nom de cada *flow* de sortida

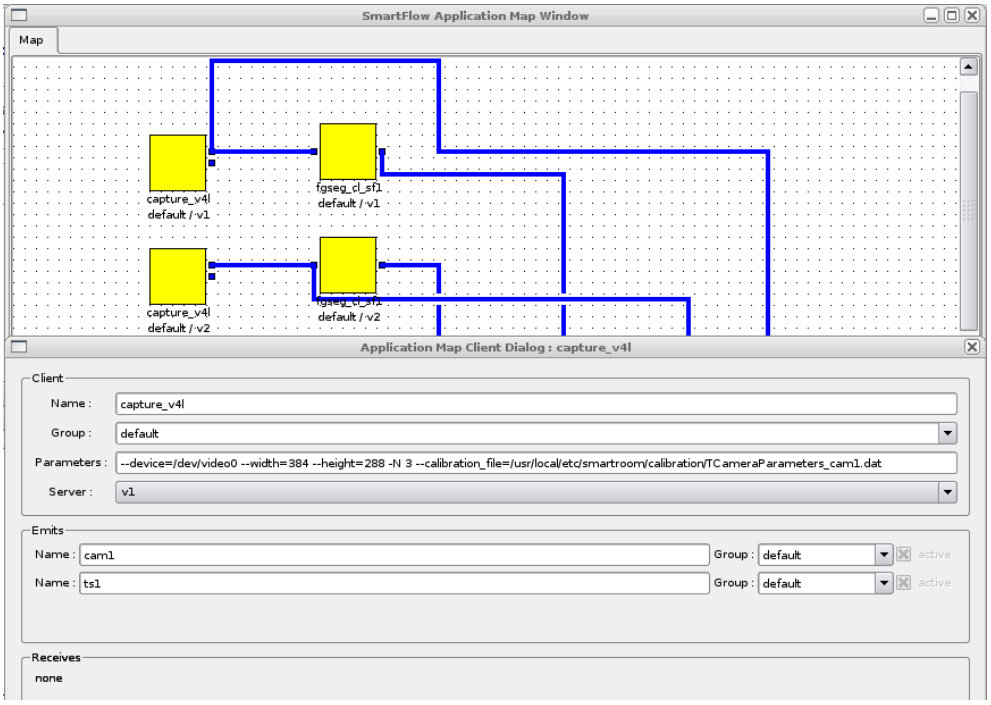


Figura 7.3: Mapa d'aplicació i característiques d'un client

La figura 7.3 mostra part d'un mapa d'aplicació així com els paràmetres de configuració d'un client. El mapa d'aplicació mostra els clients seleccionats amb caixes, unides pels *flows* que els relacionen. En els paràmetres de configuració dels client de captura s'especifica el servidor utilitzat, la mida de les imatges proporcionades, el fitxer de calibració relacionat amb les imatges i el nom dels *flows* de sortida.

## 7.3 Arquitectura

Per tal de distribuir la càrrega de treball entre els diversos servidors cal tenir en compte tant el cost computacional dels diversos clients com les connexions físiques de les càmeres amb els servidors.

El demostrador proposat treballa amb 7 càmeres i realitza la reconstrucció i la representació de la mateixa. Els clients requerits són els següents:

- 7 clients de captura de vídeo: proporcionen les imatges RGB d'entrada
- 7 clients de segmentació de primer pla: llegeixen imatges RGB i proporcionen imatges d'un sol canal amb les siluetes
- 1 client de reconstrucció: llegeix les siluetes i proporciona un *flow* amb els punts trobats i la seva orientació
- 1 client d'acoloriment dels punts: llegeix les imatges RGB i els punts trobats, proporcionant un *flow* amb els punts trobats i el color assignat a cada punt
- 1 client de representació de l'escena: renderitza la representació basada en punts en un entorn que permet la seva visualització i manipulació interactiva

Dels clients anteriorment descrits, els que suposen una càrrega més important per la *GPU* són els de reconstrucció, l'acoloriment dels punt i la representació de l'escena. El client de reconstrucció és el que realitza operacions amb un cost computacional més elevat, fent necessari utilitzar la *GPU* més potent (GeForce GTX 295). Els clients d'acoloriment de punts i representació de l'escena s'executen a la Quadro FX 3700 tant per la seva capacitat computacional com per realitzar operacions amb gràfics.

Els altres processos són menys costosos que els anteriors però cal repartir-los correctament entre les màquines per tal de balancejar el treball entre elles. Tenint en compte que dues màquines es troben ocupades amb clients amb fortes demandes computacionals es podria pensar en deixar la resta de clients a les altres màquines.

Les capturadores de vídeo es troben connectades físicament als diversos servidors. És per aquest fet que tant el client de captura com el de segmentació de primer pla s'executaran distribuïts entre les diverses màquines.

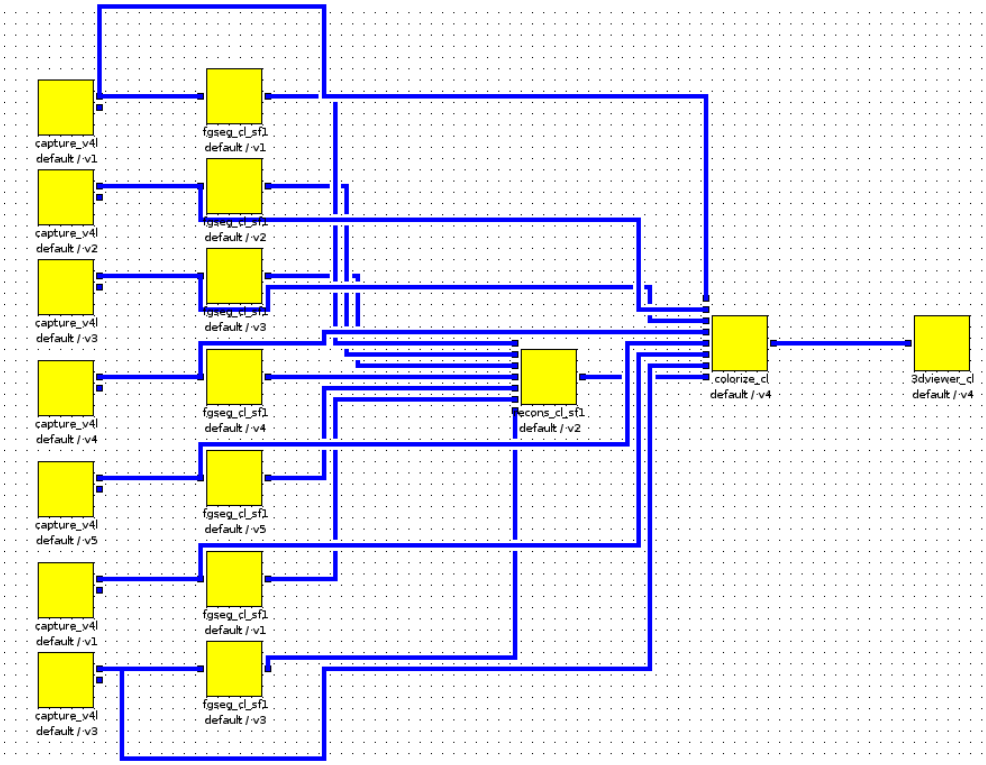


Figura 7.4: Mapa d'aplicació complet del demostrador

La figura 7.4 mostra el mapa *smartflow* complet amb els diversos clients i la màquina on s'executa cadascun. Cada màquina té un client de captura i un de segmentació de primer pla, excepte la 1 i la 3 que en tenen 2.

## 7.4 Decisions de disseny

En primer lloc es va decidir utilitzar imatges delmades a la meitat. Les imatges proporcionades per les càmeres són entrellaçades, delmant a la meitat s'aconsegueix reduir la mida de les dades, augmentant la probabilitat de trobar punts de superfície. La qualitat dels resultats obtinguts amb les imatges entrellaçades i amb les delmades és similar.

Una altra mesura va consistir en adaptar el nombre d'imatges al nombre de reconstruccions realitzades. Aquesta acció aconsegueix reduir el nombre de dades a transmetre per instant de temps, reduint la càrrega de la xarxa. La captura d'imatges per les càmeres es realitza a un ritme de 24 imatges per segon. Treballant amb aquest ritme, la xarxa es congestiona i es produeixen retards

apreciables.

El client de reconstrucció és el més lent de la cadena, podent processar 9 jocs d'imatges per segon. El nombre d'imatges per segon proporcionades per les càmeres és sempre el mateix, però el client de captura pot descartar imatges. Descartant 2 de cada 3 imatges s'aconsegueix treballar amb 8 imatges per segon, velocitat propera a les 9 imatges per segon del client més lent.

La gestió dels *flows* és interna a *smartflow* però no realitza un control entre les mostres de temps dels diversos *flows*. Aquesta absència de control provoca que en mapes on es concatenen diverses etapes es treballi amb dades incorrectes. En el demostrador presentat s'observa que en realitzar l'acoloriment les imatges utilitzades són més recents que les imatges en que s'ha realitzat la reconstrucció. Aquest problema és fàcilment visible ja que zones de la reconstrucció s'acoloreixen malament ja que, degut al moviment, l'escena pot haver canviat de configuració.

Per tal de controlar el sincronisme entre els diversos *flows* s'implementa un *buffer* circular al client d'acoloriment. Per cada reconstrucció arribada al client, es busca la imatge de cada càmera més propera a la marca temporal de la reconstrucció per tal de realitzar un acoloriment amb les mateixes imatges en que s'ha realitzat la reconstrucció.

## 7.5 Representació de l'escena

La representació de l'escena realitzada utilitza OpenGL per dibuixar la reconstrucció. La representació recrea la *smart room* utilitzant unes textures que recreen les parets i el terra de la sala. La figura 7.5 mostra la recreació de la sala. Les parets utilitzen fotografies preses a la sala, mentre que el terra és una textura d'OpenGL.

A cada *frame* es dibuixen tots els punts de la sala en la posició 3D de la representació on s'han trobat. Cada punt es representat per un cub 3D del color indicat. La mida dels cubs és ajustable, en cas de representar els cubs d'una mida massa petit la reconstrucció de superfície presentarà forats i es podrà veure a través. Mides massa grans seran apreciables si ens apropem a la reconstrucció.

En la representació realitzada no s'observen els diferents objectes estàtics que hi pugui haver a la sala. Com s'observa a 7.5 la representació de la sala és estàtica per la qual cosa no s'observen els elements que hi pugui haver a la sala durant l'execució. La reconstrucció sí que comprendria els objectes estàtics si

l'aprenentatge del fons es realitzés amb la sala totalment buida.

La implementació amb OpenGL permet una interacció amb la representació realitzada podent variar l'angle i la distància en que l'escena és visualitzada. La representació pot ser observada des de qualsevol punt de vista.

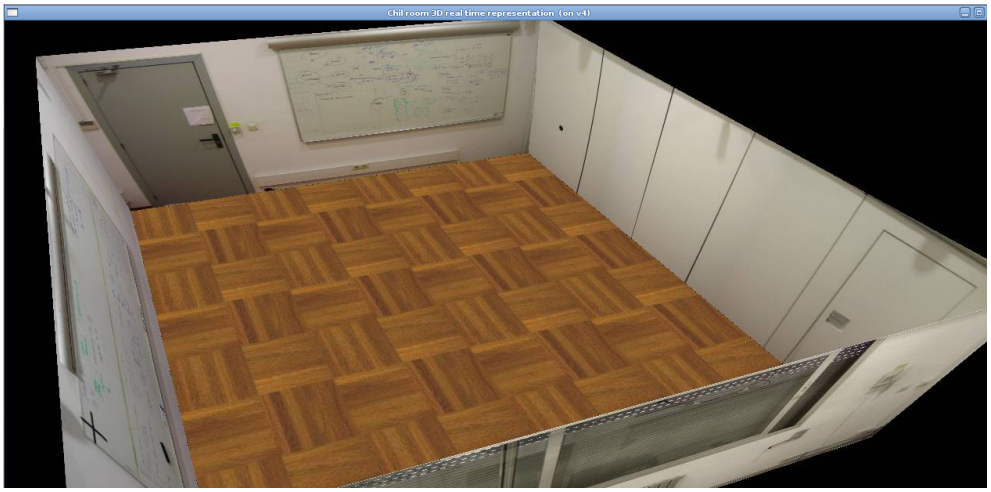

Figura 7.5: Representació de la sala buida

A més de la possibilitat de variar lliurement el punt de vista de l'escena s'han habilitat una sèrie d'opcions modificables mitjançant el teclat:

- Visualitzar l'escena des de la posició de les diverses càmeres de la *smart room*. Amb el vídeo de les càmeres es pot comparar la qualitat de la reconstrucció amb les imatges de la càmera.
- Visualitzar o no les parets i/o el terra: la recreació de la sala permet contextualitzar la reconstrucció, però puntualment pot interessar observar únicament la reconstrucció.
- Visualitzar els límits on es realitza la reconstrucció: la reconstrucció no es realitza a tota la sala, podent dibuixar els límits on es realitza
- Representar la reconstrucció d'un color uniforme: tots els punts es pinten d'un únic color
- Augmentar o disminuir la mida dels punts: permet ajustar la mida en funció de la distància en la que es visualitza la reconstrucció
- Canviar el color de fons: el color de fons de l'escena reconstruïda pot ser negre o beix

- Canviar la llum dels punts: visualitzar els elements pel color indicat o modificar-los en funció dels elements que els envolten i la posició de la llum
- Canviar les textures: canviar el dibuix de les parets i el terra per unes textures que simulen un paisatge exterior

El resultat d'aquesta representació és prou adequat als requeriments inicials del projecte: es visualitza una reconstrucció de superfície amb un *frame rate* prou elevat per tenir una percepció de continuïtat de moviment. No obstant això, la aleatorietat en el càlcul de punts i el soroll present en les normals calculades degraden la visualització. A cada instant de temps, els punts canvien de posició i la orientació dels mateixos pot ser molt diferent en punts que es trobin en una mateixa zona. Per tal de reduir aquest efecte es proposa una nova aproximació a l'acoloriment i a la representació de l'escena.

### 7.5.1 Representació de l'escena dependent de la vista

El principal problema de la representació de l'escena independentment del punt de vista permet acolorir el punt amb el seu color real si l'orientació del punt és correcta. L'orientació de cada punt determina les càmeres amb les quals s'acoloreix el punt. La reconstrucció realitzada estima les normals a partir d'un nombre de punts elevat, tot i així l'orientació és sorollosa, ja sigui per haver-hi pocs punts propers, per la distribució en bins o per que la reconstrucció no és prou densa i s'observen punts de l'altre cara.

En la visualització s'aprecia el problema amb l'orientació dels punts, ja que alguns punts de la superfície agafen colors diferents als punts propers en alguns *frames* puntuals. Per tal de reduir aquest problema es proposa integrar l'acoloriment dels punts amb el visor, acolorint els punts en funció de la posició on visualitzem l'escena.

Realitzant en un sol client l'acoloriment i la visualització de l'escena es pot determinar en cada moment la posició del visor abans d'acolorir els punts. D'aquesta manera tots els punts poden ser acolorits amb les tres càmeres més properes a la posició de la càmera virtual, sempre que cada punt es trobi en línea de visió directa amb la càmera.

Els resultats obtinguts amb aquest visor presenten una major consistència temporal, millorant la percepció obtinguda. El desavantatge d'aquesta configuració és que un sol client s'encarrega del visor i de l'acoloriment. A causa d'aquest

augment en el cost computacional del client, l'escena es refresca més lentament, reduint la sensació de continuïtat de moviment.

En el present capítol s'ha presentat la sala on es realitza el demostrador, així com la tecnologia usada per treballar amb processos repartits per diverses màquines. Finalment, s'ha introduït la forma de representar l'escena elegida.

# Capítol 8

# Conclusions i treball futur

L'objectiu d'aquest projecte final de carrera consistia en aconseguir una reconstrucció 3D de superfícies en temps real. S'ha aconseguit integrar la reconstrucció de superfícies en un demostrador a temps real. El demostrador treballa a un *frame rate* de 8 imatges per segon, apropant-se a l'objectiu de 10 reconstruccions per segon.

## 8.1 Resum de contribucions

Per tal d'assolir l'objectiu de realitzar una reconstrucció a temps real s'han realitzat una sèrie de contribucions en diverses àrees. Aquestes contribucions principalment consisteixen en adaptar algorismes seqüencials per aconseguir execucions paral·leles eficients.

**Projecció i model de càmera**
: La implementació del model de càmera en OpenCL permet corregir la distorsió de lent de les imatges en què es treballa al demostrador (384 columnes i 288 files) amb una reducció del temps d'execució de 60 cops. La projecció permet la relació d'un punt de l'espai 3D amb les coordenades d'imatge, que utilitzarem en la reconstrucció

**Segmentació de primer pla**
: La segmentació de primer pla i el seu post-processament amb morfologia matemàtica permet generar les siluetes necessàries en la reconstrucció. La implementació paral·lelitzada redueix a una dotzena part el temps d'exe-

cució en front de la implementació seqüencial en les dimensions d'imatge utilitzades

**Reconstrucció 3D de superfície**
S'ha realitzat un mètode de generació de reconstrucció de superfícies que assoleix una qualitat suficient per la seva visualització en temps real. S'ha assolit un nombre d'entre 8 i 10 reconstruccions per segon amb les condicions requerides. Augmentant la implementació seqüencial entre 2 i 5 cops, depenent si es considera el post-processament seqüencial o no

**Acolorir els punts**
Amb l'ajuda de mapes de profunditat creats amb OpenGL s'acoloreixen els punts. El seu temps d'execució depèn del nombre de punts trobats, però en cap moment és superior al temps d'execució de la reconstrucció de superfície

**Demostrador**
Els diversos elements s'han integrat en un demostrador en temps real. S'han realitzat dues alternatives de visualitzar l'escena: una acolorint els punts únicament amb la informació de l'orientació de cada punt i una segona acolorint els punts en funció de la posició de la càmera virtual

**Introducció OpenCL**
El projecte final de carrera realitzat ha servit per introduir OpenCL al Grup de Processament d'Imatge i Vídeo, proporcionant un entorn i una metodologia per treballs futurs

## 8.2 Treball futur

El present projecte aconsegueix l'objectiu d'aconseguir una metodologia de reconstrucció de superfície en temps real. A partir d'aquest treball sorgeixen diferents línees de millora.

**Augmentar la zona de treball**
S'ha aconseguit realitzar una reconstrucció de superfície en un espai de $8m^3$. Un augment de l'espai de cerca a tota la *smartroom* seria una millora a considerar. Es podria realitzar amb una targeta gràfica més potent o dividint el processament en etapes. Una *GPU* podria realitzar una voxelització per tot l'espai i una segona podria buscar els punts de superfície

**Portar el sistema complet a altres entorns**

El desenvolupament de l'algorisme ha tingut en compte els resultats obtinguts a la *smartroom* de la UPC, considerant també altres configuracions amb diferent nombre de càmeres i píxels en cada imatge amb bons resultats. Tot i aquest treball en diferents configuracions, seria interessant observar el sistema complet en una sala diferent. La nova sala intel·ligent que s'està construint a la UPC pot ser un bon entorn per portar-ho a terme al comptar amb càmeres amb més resolució, nous servidors i un espai més indicat per desenvolupar els algorismes

**Ús d'altres sensors**

Utilitzant el mètode presentat amb càmeres amb major nombre de píxels i major qualitat d'imatge permetria augmentar la qualitat de la reconstrucció però possiblement augmentaria el temps de procés. Amb l'ús de nous sensors que proporcionen la profunditat a l'escena com són les kinect, l'aproximació al problema canviaria radicalment

**Millora de la segmentació de primer pla**

La segmentació de primer pla realitzada obté siluetes eficientment, utilitzant mètodes de segmentació més elaborats es podria augmentar la qualitat de la reconstrucció

**Millora en l'estimació de l'orientació dels punts**

El mètode d'estimar l'orientació dels punts és molt sorollós, provocant una degradació al suavitzat de la superfície i a l'acoloriment dels punts. Una millor estimació de les normals hauria de tenir en compte els bins adjacents a l'hora de buscar veïns

**Reconstrucció com a base de noves anàlisis**

La reconstrucció realitzada pot esdevindre la base de nous algorismes en temps real, com poden ser el seguiment dels elements de l'escena o adaptar un model del cos humà a la reconstrucció realitzada

# Apèndix A

# Entorn programació OpenCL

La *API* d'OpenCL es troba disponible tant per a C com per a C++. L'entorn de desenvolupament del Grup de Processament d'Imatge i Vídeo utilitza C++ com a llenguatge de programació, podent integrar l'*API* per a OpenCL.

Per tal d'integrar OpenCL a l'entorn de desenvolupament s'ha optat per crear una classe -ContextGPU- que encapsula les crides a la *API* d'OpenCL. El ContextGPU s'encarrega de tota la comunicació entre el *host* i els dispositius, ja sigui establir el context comú, crear les estructures de memòria i les funcions necessàries o controlar l'execució als dispositius.

El ContextGPU permet reduir els paràmetres requerits a la majoria de crides a la *API* al tenir uns paràmetres per defecte que en la *API* s'han d'introduir cada cop. Per exemple, a la *API* per escriure en un buffer s'ha d'indicar si l'operació és bloquejant (s'espera a acabar l'escriptura abans de seguir amb l'execució), en quina posició es comença a escriure, quina mida s'escriu i un punter a les dades. Usualment voldrem inicialitzar el buffer sencer de forma bloquejant per tant la posició, la mida i el *flag* bloquejant es troben per defecte.

El ContextGPU no només conté la interfície amb OpenCL, també presenta una sèrie d'utilitats per facilitar la programació amb OpenCL. Pel que fa a l'estructura dels fitxers a programar s'han declarat dues utilitats:

- *stringfromcode*: macro que permet declarar el codi OpenCL còmodament. El codi OpenCL es declara al *host* com un *string* i es compilat en temps d'execució. La macro *stringfromcode* permet declarar diferents strings OpenCL en un sol fitxer, podent elegir quins d'ells són utilitzats pel ContextGPU còmodament

- *stringandcode*: per tal de compartir estructures de dades entre *host* i els dispositius OpenCL és necessari declarar les estructures en les dues interfícies. La macro *stringandcode* possibilita evitar duplicar el codi ja que per una banda declara el codi C++ i per altra declara el *string* amb el codi OpenCL

Per tal de poder definir estructures compartides amb la macro stringandcode s'ha de definir un alineament de les estructures comú a OpenCL i a c++. Declarant les estructures amb l'atribut *packed* s'ha comprovat que l'alineament de les dades és el mateix i que per tant es possible transferir paràmetres entre *host* i els dispositius.

Un exemple d'ús de la utilitat *stringandcode* juntament amb l'atribut *packed* es pot veure a continuació:

```
const CodeGPU cl_src_camera_cl=__string_and_code(

 struct __attribute__((packed)) TCameraCL
 {
        //!projection matrix
        cl_float proj[3*4];
        //!calibration matrix
        cl_float cal[3*3];
        //!distortion parameters
        cl_float distortion[4];
        //!width of the image
        cl_uint im_width;
        //!height of the image
        cl_uint im_height;
 };
 typedef struct TCameraCL CameraCL;
);
```

El ContextGPU defineix també codi OpenCL amb funcionalitats homogeneïtzadores:

1. La *API* defineix les variables amb diferent nom a la interfície host respecte als dispositius: al host es defineixen amb el prefix cl_ (cl_int) mentre que a la interfície OpenCL es defineix sense (int). OpenCL presenta vectors de les diferents variables amb accessos específics als elements. Per exemple:

   cl_int4 (int4 a OpenCL) es tracta d'un vector de 4 enters.

l'accés als diferents elements es realitza amb sufixos (.x, .y, .z i .w) Aquesta notació (int4 o int16) pot donar a equívocs ja que usualment el número després de la variable indica el nombre de bits que conté.

2. Per evitar equívocs i la diferència de notació entre les dues interfícies, es defineixen les variables en OpenCL com la notació en el *host*. Es recomana usar sempre la notació *host* (utilitzar cl_intX cl_uintX cl_floatX en lloc de intX uintX floatX, amb X qualsevol mida acceptat per la API (cap, 2, 3, 4, 8, 16).

- La funció not_valid_thread s'ha definit per tal d'evitar execucions de *work-items* fora del *global-group*. Si el *work-item* forma part del *work-group* es permet l'execució del *work-item*, d'altra forma el *work-item* no ha de seguir l'execució
- Diferents funcions s'han definit per accedir a estructures de dades de diferents dimensions. Una organització dels *work-items* en dues o tres dimensions pot voler accedir a un buffer d'una dimensió.

# Apèndix B

# Orientació dels punts

El càlcul de normals és necessari per tal de disposar d'una forma de calcular una base de l'espai donats una sèrie de punts. Interessarà especialment disposar del vector ortogonal a una sèrie de dades ja que aquest vector ens indicarà l'orientació dels punts de superfície. Un cop calculat el vector ortogonal el càlcul dels altres dos vectors de la base és immediat.

$$A = \sum_{i=1}^{neighbors} \begin{pmatrix} x_i x_i & x_i y_i & x_i z_i y_i x_i & y_i y_i & y_i z_i z_i x_i & z_i y_i & z_i z_i \end{pmatrix}$$

Un cop generada la matriu es calcula una base de l'espai: els dos autovectors associats als autovalors de més energia formen l'espai de les dades i el autovector amb menor energia (autovalor menor) correspon al vector ortogonal a les dades.

# Apèndix C

# Generació de nombres aleatoris

L'estàndard d'OpenCL no disposa d'una funció per tal de generar nombres aleatoris. Per tal de generar nombres aleatoris s'ha adaptat un generador presentat a *GPUGEMS*, una revista de NVIDIA amb articles sobre la programació en *GPUs*.

L'article proposa combinar tres *Linear congruential generator* (LCG) amb un generador Tausworthe. D'aquesta forma s'aconsegueix un període més gran que els d'un generador pseudo-aleatori sol (període combinat de l'ordre de $2^{121}$) i es redueix la correlació entre les mostres, greu problema tant pels generadors LCG com pels Tausworthe.

El generador LCG es un generador pseudo-aleatori definit per la següent relació de recurrència:

$X_{n+1} = (aX_n + c)(modm)$

amb m, a, c, $X_0$ valors enters constants: m, m > 0 - el mòdul a, 0 < a < m - el multiplicador c, 0 ≤ c < m - l'increment $X_0$, 0 < $X_0$ < m - la llavor o valor d'inici

El generador Tausworthe és un generador multiplicatiu recursiu que produeix bits aleatoris:

$x_{n+1} = (A_1x_n + A_2x_{n-1} + ... + A_kx_{n-k+1})mod2$

amb $x_i$, $A_i \in 0, 1 \; \forall$ i

# Bibliografia


[1] Garcia, O. Mapping 2D images and 3D world objects in a multicamera system. Ms. Thesis. Technical University of Catalonia. 2004.

[2] Hartley, R i Zisserman, A. Multiple view geometry in computer vision. Cambridge University Press, 2000.

[3] Piccardi, M. Background subtraction techniques: a review, in Proc. of IEEE SMC 2004 International Conference on Systems, Man and Cybernetics, The Hague, The Netherlands, October 2004.

[4] Serra, J. Curs de morfologia matemàtica. http://cmm.ensmp.fr/ serra/-cours/index.htm

[5] Laurentini, A. (1994). The visual hull concept for silhouettebased image understanding. IEEE Trans. on Pattern Analysis and Machine Intelligence.

[6] Kutulakos, N. i Seitz, M. (1999). A theory of shape by space carving. In Proc. IEEE Int. Conf. on Computer Vision, volume 1, pages 307 314.

[7] Salvador, J. Surface Reconstruction for Multi-View Video Analysis. Ms. Thesis. Technical University of Catalonia. Setembre 2011.

[8] E. Oriol. Algoritmos de reconstruccin de escenas 3D paralelizados en GPU para aplicaciones en tiempo real, Universitat Politècnica de Catalunya. 2010.

[9] Friedman, J. H., Bentley, J. L., i Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. ACM Trans. on Mathematical Software, 3(3):209-226.

[10] Howes L. i Thomas D. NVIDIA GPU Gems 3 - Chapter 37. Efficient Random Number Generation and Application Using CUDA.

[11] Grup Khronos. http://www.khronos.org/