UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

VISUALITZACIÓ DE NÚVOLS DE PUNTS DE SUPERFÍCIES

Autor: Eudald Elias Vilanova

Directora: Dra. Anna Puig
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 24 de gener de 2022

Abstracte

La visualització d'un núvol de punts és la generació de visualitzacions a partir d'una mostra discreta i finita d'un objecte 3-dimensional. Un dels tipus més comuns de núvols de punts, són mostres discretes de superfícies, que s'obtenen, per exemple, a partir d'escàners amb làser del món real (*LIDAR*). La utilitat dels núvols de punts pot ser variada, des d'anàlisis de l'estructura del terreny (en l'agricultura), reconeixement d'objectes, inspeccions mèdiques, etc. Però el que sempre cal és una forma de generar visualitzacions del núvol de punts perquè aquests puguin ser estudiats per a persones.

Aquest treball presenta un mètode de visualització específicament per a mostres discretes de superfícies de l'espai, anomenat *Splatting*. El mètode no només es limita en generar imatges digitals, sinó que pretén oferir una visualització interactiva del núvol de punts, és a dir, la generació d'imatges ha de ser prou ràpida per a aconseguir un efecte de visualització en temps real.

2010 Mathematics Subject Classification. 65S05

Agraïments

Gràcies a l'Anna Puig, la directora del meu treball.

Índex

Introducció

Context

La **visualització** (o **renderització**) **3D** és el procés de creació d'una **imatge digital** representativa d'una **escena** o **model** tridimensional virtual mitjançant l'ús d'un ordinador. Un **model 3D** és una representació matemàtica de qualsevol objecte tridimensional, incloenthi la forma geomètrica i propietats perceptives com el **color** o **material** del objecte. D'altra banda, una **escena 3D** és una agrupació d'un o diversos models 3D juntament amb informació de possibles fonts d'il·luminació i el punt de vista de la visualització.

La renderització és la branca principal de la *Computació Gràfica* pertanyent a les *Ciències de la Computació*. Però, d'igual forma que passa en diversos problemes de la computació, el procés de renderització conté conceptes que provenen d'altres disciplines, com per exemple, les *Matemàtiques* (projeccions geomètriques, transformacions lineals, interpolacions, coordenades baricèntriques, plans tangents, etc.) o la *Física* (models de reflexió, refracció i dispersió de la llum, propietats dels **materials**, etc.).

Aquí es planteja el problema de visualització de **núvols de punts** de **superfícies**. Tot i que les superfícies s'estudien en els àmbits de la *Topologia* i *Geometria*, els núvols de punts són mostres discretes de superfícies, i, per tant, també s'aplicaran conceptes de *Matemàtica Aplicada* i *Estadística* (reconstrucció de models continus a partir de mostres discretes).

Motivació

A causa de la proliferació de vehicles i robots autònoms, la visió artificial i en concret el reconeixement d'objectes està essent un àmbit de la computació en plena expansió. La forma en què els vehicles i robots escanegen el seu entorn (mètode *LIDAR* [19], veure Figura 1), genera dades tridimensionals en forma de núvol de punts de superfícies. Aquestes dades, un cop generades, s'envien als algorismes de reconeixement per detectar objectes d'interès.

Un factor clau pel desenvolupament de programes de reconeixement d'objectes en núvols de punts és l'habilitat de visualitzar i explorar aquests núvols de punts de forma

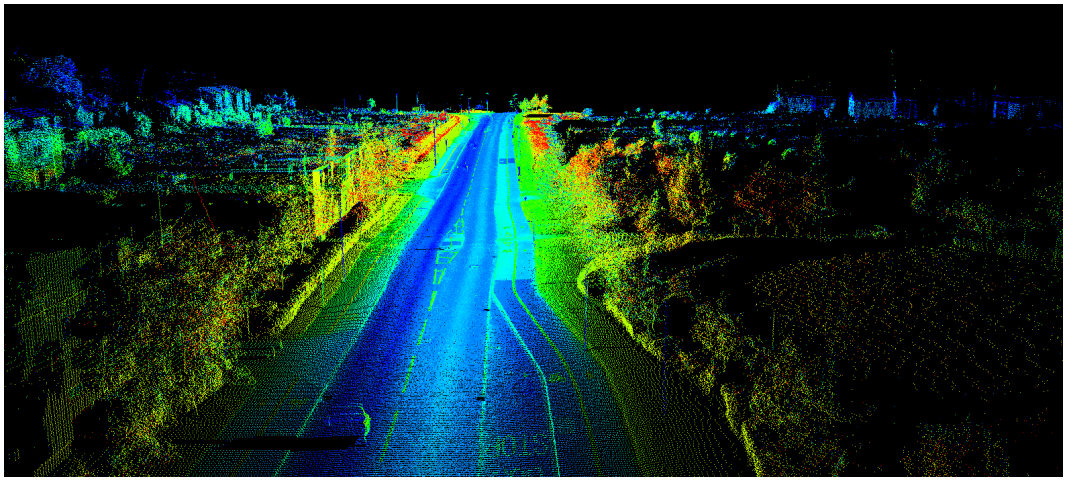


Figura 1: Exemple d'un núvol de punts generat amb el mètode *LIDAR*[19]

interactiva, apreciant-ne les característiques geomètriques, i podent contrastar les parts que han sigut reconegudes o segmentades pels algorismes.

Però el problema del reconeixement d'objectes és, tan sols, un cas concret de molts altres que requereixen visualitzacions de núvols de punts de superfícies. Altres exemples podrien ser: l'anàlisi del terreny per a estudis geològics de zones, estudis anatòmics d'imatges mèdiques, etc. Per tant, podem afirmar que oferir una bona visualització per a núvol de punts és essencial pel desenvolupament humà en molts àmbits.

Hi ha vàries maneres de visualitzar núvols de punts. Una d'elles és considerar cada punt com a una figura geomètrica primitiva, per exemple una esfera d'un radi determinat, i generar la imatge d'aquesta interpretació del model. Una altra manera més elaborada és aplicar un mètode de reconstrucció a la mostra discreta (com per exemple una **convolució**) per tal de generar un nou model 3D que s'aproximi al model original (del qual s'ha extret la mostra), i llavors visualitzar aquest nou model. Les visualitzacions amb reconstrucció ofereixen una millor interpretació geomètrica de les superfícies, i com a tal, en aquest projecte, es pretén estudiar i elaborar un algorisme per visualitzar una reconstrucció d'un núvol de punts de forma interactiva.

Objectiu General

L'objectiu és realitzar l'estudi i desenvolupament d'una estratègia que, donada una mostra discreta i finita (**núvol de punts**) d'una superfície, **visualitzi** en **temps real** [2] una reconstrucció de la superfície original. A més, també es vol oferir un efecte d'il·luminació bàsic aplicant un **ombrejat local** en cada punt de la reconstrucció.

[2]Pel que fa al significat de temps real, s'entendrà que la freqüència de refresc de la visualització sigui suficient per tenir l'efecte de moviment, per exemple una freqüència de 30 FPS (*frames per second*, de l'anglès, imatges per segon) serà acceptable

Objectius Concrets

EL objectius específics d'aquest projecte són:

1) **Anàlisi i estudi de mètodes de visualització de núvols de punts.**

2) **Formalització del problema.**

3) **Anàlisi, disseny i implementació d'un entorn interactiu de simulació per a diferents mètodes de visualització de núvols de punts.**

4) **Anàlisi, disseny i implementació d'ombrejat local.**

5) **Elaboració i validació dels resultats.**

Tasques

Els objectius concrets es poden desglossar en les tasques següents:

1) **Creació d'un entorn interactiu**: S'ha de crear l'estructura bàsica del codi; configurar l'entorn gràfic del sistema; lectura de *inputs* de l'usuari; definir estructura d'una *càmera* i mètodes per controlar-la; i el bucle principal del programa que iterarà cada visualització, actualitzant les dades a partir dels *inputs* de l'usuari.

2) **Visualització directa de núvols de punts**: Definir l'estructura d'un núvol de punts i funcions bàsiques per manipular-lo; crear la lectura d'un fitxer de núvol punts; crear el primer mètode de visualització per a punts on cada punt és una simple esfera.

3) **Visualització de núvols de punts amb reconstrucció**. En aquesta part s'ha de crear un nou mètode de visualització que apliqui una funció de *convolució* per a aproximar la superfície original.

4) **Aplicació d'ombrejat local**. Caldrà introduir l'estructura d'una font d'il·luminació, així com el càlcul de la reflexió de llum en un punt de la superfície. No es tindran en compte les ombres creades pel bloqueig de llum en parts còncaves.

5) **Mètodes d'optimització per reduir el temps de visualització**. Desenvolupaments d'estratègies que reduiran el cost de visualització sovint a canvi de menor fidelitat.

6) **Simulacions, resultats i validacions del mètodes desenvolupats**

7) **Elaboració de la memòria**

Planificació

Tasca	Agost	Setembre	Octubre	Novembre	Desembre	Gener
Preparació						
Desenvolupament						
Creació d'un entorn interactiu						
Visualització directe						
Visualització amb reconstrucció						
Ombrejat local						
Optimització						
Simulacions i resultats						
Documentació						

Taula 1: Diagrama de Gantt

Organització del document

L'estructura d'aquesta memòria es desglossa en els següents capítols:

- **Introducció:** Es posa el context del projecte i s'esmenten els objectius i tasques a dur a terme.
- **Formalització del problema:** En aquest capítol es defineix els conceptes necessaris per entendre el problema, així com les característiques que ha de complir una solució.
- **Mètodes de visualització**: Es presenten solucions conegudes a problemes similars per tal de, primer extreure'n idees i metodologies aplicables en el nostre cas i segon obtenir una base comparativa amb el nostre mètode.
- **Solució teòrica**: Aquí es detallen els processos i algorismes teòrics per tal de dur a terme una visualització amb reconstrucció de núvols de punts.
- **Disseny i desenvolupament**: Aquí es detalla l'anàlisi, disseny i tècniques emprades per tal de crear el programa de visualització amb els requisits necessaris, utilitzant els algorismes explicats a la solució teòrica.
- **Resultats i Simulacions:** Es mostren els resultats del projecte desenvolupat. Incloent comparatives visuals de diferents mètodes i programes.
- **Conclusions i feina futura:** Es detallen les conclusions i l'estat del projecte de cara al futur.

- **Apèndix: Manual Tècnic:** Es descriuen els requisits i passos per instal·lar i usar el programari.

Capítol 1

Formalització del problema

1.1 Introducció

Aquest capítol servirà per introduir els conceptes que formen part d'un procés de visualització de núvol de punts en general, i definir el problema concret i objectius que es proposen en aquest treball.

En general, una **visualització 3D** és el procés de generar una **imatge digital** que representi o simuli la **visió** i **il·luminació** d'un **model 3D** a través d'un observador o **càmera**. Abans d'entendre com funciona una visualització, es necessari introduir els conceptes que hi juguen un paper fonamental. Aquests són:

- Models 3D de superfícies i núvols de punts
- Reconstruccions de mostres discretes
- Colors digitals i il·luminacions
- Projeccions de càmera

Per donar una idea intuïtiva, en la Figura 1.1, es mostra un diagrama d'un procés de visualització per a un núvol de punts.

1.2 Superfícies

Hi ha diverses formes de definir el concepte de superfície depenent de l'àmbit d'estudi. Pel problema de visualització, és d'interès estudiar superfícies de $\mathbb{R}^3$ que representin objectes del món real. Aquest tipus de superfícies, es defineixen matemàticament com a **superfícies diferencials**.

Però els ordinadors no poden representar conjunts densos de dominis continus com $\mathbb{R}^3$, ja que tenen memòria finita. Com a tal, les superfícies se solen representar en ordi-

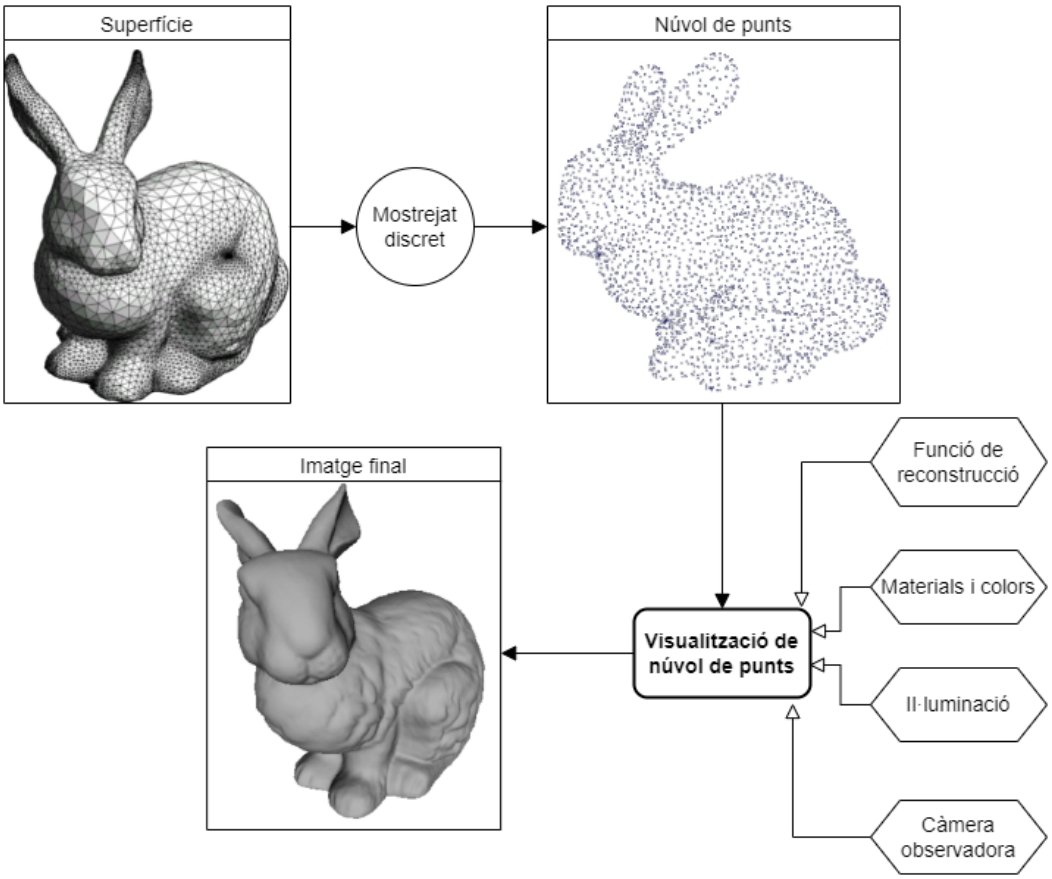


Figura 1.1: Diagrama conceptual d'una visualització d'un núvol de punts

nadors com a una col·lecció discreta de vèrtex, connectats en triangles, aquest tipus de superfícies es coneix com a una **malla de triangles**. Una malla de triangles no és directament una superfície diferencial (com es veurà), no obstant, sí que és una superfície en el sentit topològic.

Primer es donarà la definició de **superfície topològica**; tot seguit la definició de **superfície diferencial**; i per acabar es definirà el concepte de **malla de triangles**, i com es pot utilitzar per representar superfícies diferencials.

1.2.1 Superfícies topològiques

Definició 1.1. Una **superfície** de $\mathbb{R}^3$ és un conjunt $S \subset \mathbb{R}^3$ amb la mètrica euclidiana de $\mathbb{R}^3$, que és localment homeomorf a un disc obert. És a dir, per a tot punt $p \in S$, existeix un obert U amb $p \in U$, i existeix una funció $\phi\colon U \to D$, on $D = \{(x, y) \mid x^2 + y^2 < 1\}$, tal que

- ϕ és bijectiva
- ϕ és contínua
- la inversa ϕ^{-1} és contínua

1.2.2 Superfícies diferenciables

El següent contingut s'ha extret dels apunts [13] de l'assignatura de *Geometria diferencial de corbes i superfícies (2020)* impartida per Ignasi Mundet i Riera del grau de Matemàtiques de la *Universitat de Barcelona*.

Definició 1.2. Una **superfície parametritzada diferenciable** és una aplicació diferenciable

$$\phi\colon \Omega \to \mathbb{R}^3$$

on $\Omega \subset \mathbb{R}^2$ és un obert (veure Figura **??**). La **traça** de la superfície parametritzada ϕ és el subconjunt $\phi(\Omega) \subset \mathbb{R}^3$.

Definició 1.3. Superfície parametritzada regular. Sigui $\phi\colon \Omega \to \mathbb{R}^3$ una superfície parametritzada. Es diu que ϕ és **regular** al punt $p \in \Omega$ si les derivades parcials

$$\phi_u(p) := \frac{\partial \phi}{\partial u}(p) \qquad \phi_v(p) := \frac{\partial \phi}{\partial v}(p)$$

són linealment independents.

Es diu que la superfície parametritzada ϕ és **regular** si ϕ és regular en tots els punts de Ω.

Definició 1.4. Pla tangent, vector normal i recta normal. Sigui $\phi\colon \Omega \to \mathbb{R}^3$ una superfície parametritzada i $p \in \Omega$ un punt on ϕ sigui regular:

- El **pla tangent vectorial** de ϕ al punt p és el subespai vectorial de $\mathbb{R}^3$ de dimensió 2 engendrat per $\phi_u(p)$ i $\phi_v(p)$ (que són linealment independents ja que ϕ és regular en el punt p). Es denota per $T_p\phi$.
- El **pla tangent afí** de ϕ al punt p és el pla afí $\phi(p) + T_p\phi$. Es denota per $T_p^{afi}\phi$.
- Un **vector normal** de ϕ al punt p és un vector perpendicular al pla $T_p\phi$. Sí el vector és unitari s'anomena **vector normal unitari**.
- La **recta normal** de ϕ al punt p és la recta afí que passa per $\phi(p)$ i és perpendicular al pla tangent afí $T_p^{afi}\phi$.

A la Figura 1.2, es mostra el pla tangent i la recta normal per a un punt d'una superfície.

Definició 1.5. Superfície regular. Sigui S un subconjunt de $\mathbb{R}^3$. Una **carta** de S és una superfície parametritzada $\phi\colon \Omega \to \mathbb{R}^3$ que satisfà les següents propietats:

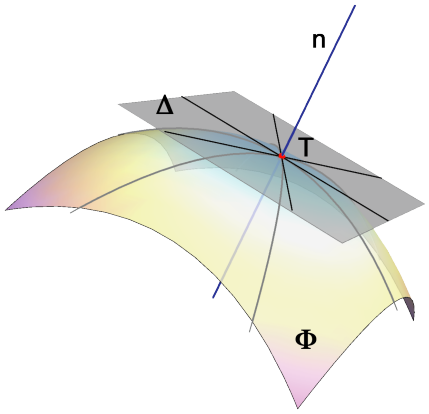


Figura 1.2: El pla tangent Δ i la recta normal n de la superfície Φ al punt T.

1. la traça $\phi(\Omega)$ està continguda dins de S i és un obert de S, és a dir, existeix un obert $V \subseteq \mathbb{R}^3$ tal que $\phi(\Omega) = S \cap V$
2. ϕ indueix un homeomorfisme $\Omega \to$, on $\phi(\Omega)$ està dotada de la topologia subespai induïda per la topologia euclidiana a $\mathbb{R}^3$
3. ϕ és una superfície parametritzada regular.

Es diu que S és una **superfície regular** si per a tot $q \in S$ existeix una carta de S, $\phi\colon \Omega \to S$, tal que $q \in \phi(\Omega)$.

Per tant, si S és superfície regular, existeix un recobriment per cartes de S. És a dir, existeix un conjunt $\{\phi_i\colon \Omega_i \to S \mid i = 1, \ldots, n\}$ tal que ϕ_i és carta de S, i

$$S = \bigcup_{i=1}^{n} \phi_i(\Omega_i)$$

Lema 1.6. Sigui S una superfície i sigui $q \in S$ un punt. Sigui $\phi\colon \Omega \to S$ i $\psi\colon \Theta \to S$ dues cartes de S, tal que $q \in \phi(\Omega) \cap \psi(\Theta)$. Aleshores, si denotem per $p_1 \in \Omega, p_2 \in \Theta$ tal que $\phi(p_1) = q$ i $\psi(p_2) = q$, tenim que

$$T_{p_1}\phi = T_{p_2}\psi$$

és a dir, el **pla tangent** d'un punt d'una superfície, no depèn de la carta escollida.

Aquest lema permet redefinir que els conceptes de la definició **??** de forma intrínseca a una superfície S, sense que depenguin de la carta escollida.

Si S és una superfície regular, aleshores cada punt té dos possibles vectors normals unitaris diferents, amb direcció oposada. Aquestes dos possibilitats del vector normal genera el concepte de **orientabilitat**.

Definició 1.7. Orientació. Sigui $S \subset \mathbb{R}^3$ una superfície regular. Una **orientació** de S és una aplicació diferenciable (i per tant, contínua)

$$N\colon S \to \mathbb{R}^3$$

tal que, per a qualsevol $q \in S$, $N(q)$ és un vector normal unitari de q.

Es diu que una superfície S és orientable si admet alguna orientació.

1.2.3 Superfícies de malles de triangles

Definició 1.8. Malla de triangles. Sigui $p_1, p_2, p_3 \in \mathbb{R}^3$, denotem per $T(p_1, p_2, p_3)$ el triangle format per els tres punts.

Sigui $(T_i)_{i=1,\dots,n}$ una col·lecció de triangles amb $n \in \mathbb{N}$, que satisfaci

$$\bigcap_{i=1}^{n} \mathring{T}_i = \emptyset,$$

on $\mathring{T}$ denota l'interior obert d'un triangle, aleshores la **malla de triangles** de $(T_i)_n$ és la superfície generada per la unió dels triangles:

$$S := \bigcup_{i=1}^{n} T_i$$

Una malla de triangles no és una superfície diferenciable, però, tot i així, es poden definir vectors normals en cada punt de la malla. Per a un triangle T, format pels punts (p_1, p_2, p_3), es defineix el vector normal unitari canònic de T com

$$\vec{n}_T := \frac{(p_2 - p_1) \times (p_3 - p_1)}{\|(p_2 - p_1) \times (p_3 - p_1)\|}.$$

Aleshores, per a qualsevol punt d'una malla de triangles, $q \in S$, es pot donar un vector normal per a q de la següent manera:

- Si $q \in \mathring{T}_i$ per algun T_i (el qual serà únic en cas que existeixi), aleshores $\vec{n}_q = \vec{n}_T$
- Si q pertany a la frontera d'algun triangle, aleshores sigui $T_1, \dots, T_k$ tots els triangles que contenen q, es dona el vector normal de q com

  $$\vec{n}_q = normalize(\frac{\sum_{j=1}^{k} \vec{n}_{T_j}}{k})$$

  el vector normalitzat de la mitjana de totes les normals.

1.3 Núvols de punts

Definició 1.9. Núvol de punts. S'anomena **núvol de punts** a una col·lecció de punts $P := (p_i)_{i=1,\dots,n}$ amb $n \in \mathbb{N}$ i $p_i \in \mathbb{R}^3\ \forall i$. En molts casos, però, cada punt d'un núvol

tindrà associat alguns elements d'uns conjunts qualsevol, com podria ser un vector, un color, un nombre identificatiu, etc. En aquests casos es referirà al núvol de punts com a una col·lecció de tuples $P := (p_i, x_i^1, \ldots, x_i^j)_{i=1,\ldots,n}$ on $p_i \in \mathbb{R}^3$ i $x_i^j \in \mathcal{X}^j$. $\mathcal{X}^j$ representen conjunts genèrics, que es definiran per a cada núvol de punts concret.

Les Figures 1.3 i 1.4 mostren exemples de núvols de punts.

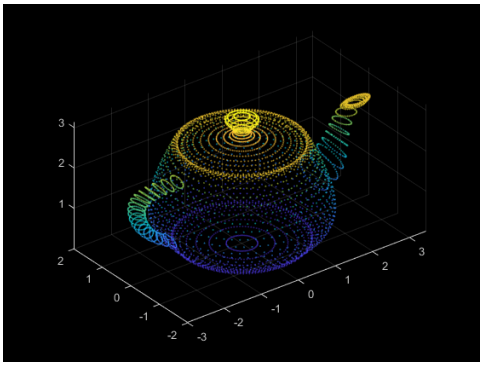


Figura 1.3: Núvol de punts

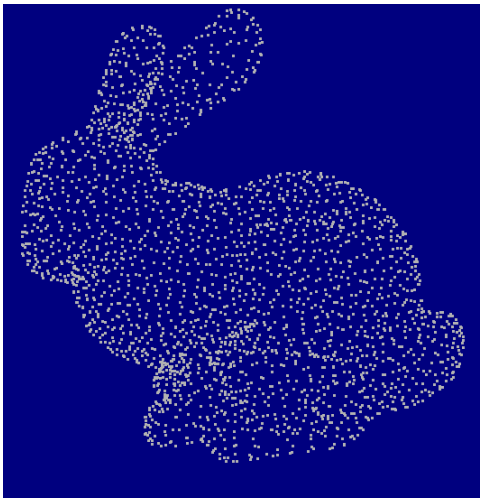


Figura 1.4: Núvol de punts

Definició 1.10. Mostreig regular. Sigui $P = (p_i)_n$ un núvol de punts, es diu que P és un núvol de punts regular, si els punts p_i segueixen una distribució equidistant. És a dir, si existeix un conjunt

$$G = \{a(i,\, j,\, k) + c\colon\ i, j, k \in \mathbb{Z}\}$$
$$a \in \mathbb{R}, c \in \mathbb{R}^3,\ \text{tal que } P \subset G$$

Definició 1.11. Mostreig aleatori. Sigui $P = (p_i)_n$ un núvol de punts, es diu que P és un núvol de punts aleatori, si els punts p_i segueixen una distribució aleatòria dins d'un espai, de forma independent. Matemàticament, una col·lecció de punts distribuïts de forma aleatòria dins d'un espai on cada punt és independent de la resta, es coneix com un *procés de punts de Poisson*. Rep aquest nom perquè el nombre de punts que es troben dins d'una regió finita de l'espai, segueix una variable aleatòria amb distribució de *Poisson*.

Les figures 1.5 i 1.6 mostren els dos tipus de mostreig.

Definició 1.12. Distància veïna. Donat un núvol de punts $P = (p_i)_{i=1,\ldots,n}$ i el seu conjunt, sigui un punt $q \in \mathbb{R}^3$, qualsevol, definim la **distància** del punt q al núvol P

$$\mathrm{d}_P(q) := \min_{i=1,\ldots,n} \|p_i - q\|$$

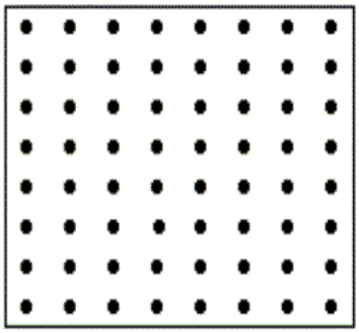


Figura 1.5: Mostreig regular

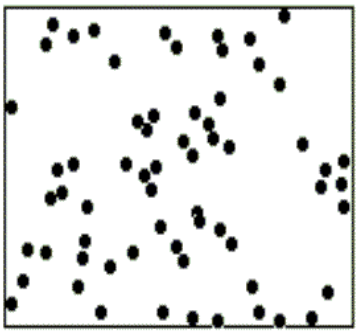


Figura 1.6: Mostreig aleatori

Sigui un punt del núvol $p \in P$, anomenem **distància veïna** del punt p al núvol P a

$$\mathrm{dv}_P(p) := \mathrm{d}_{P\setminus\{p\}}(p),$$

la distància del punt més proper del núvol diferent d'ell mateix.

Definició 1.13. K Veïns pròxims. Donat un núvol de punts $P = (p_i)_{i=1,\dots,n}$, sigui un punt $q \in \mathbb{R}^3$ i un $k \in \mathbb{N}$ s'anomenen els **k veïns pròxims** de q del núvol P, com el conjunt ordenat:

$$\mathrm{V}_P(q,k) = \{p_{j_1}, \dots, p_{j_k} \mid \mathrm{d}(q,p_{j_1}) \leq \dots \leq \mathrm{d}(q,p_{j_k}) \quad j = 1,\dots,k\},$$

on $\mathrm{d}(p,q) := \|p-q\|$

Definició 1.14. Entorn de punts. Donat un núvol de punts $P = (p_i)_{i=1,\dots,n}$, sigui un punt $q \in \mathbb{R}^3$ i un $r > 0$, s'anomena l'**entorn de punts** de radi r i centre q del núvol P, com el conjunt:

$$\mathrm{B}_P(q,r) = \{p \in P \mid \|p-q\| \leq r\}$$

1.4 Convolució

En aquesta secció s'explicarà l'operació de convolució de funcions [24]. Aquesta operació serà la tècnica que farem servir per reconstruir un núvol de punts en un model continu. Primer es definirà la convolució per a funcions en $\mathbb{R}$, i després es definirà el concepte de convolució per a núvols de punts.

1.4.1 Convolució de funcions reals

Definició 1.15. Sigui $f, g\colon \mathbb{R} \to \mathbb{R}$ dos funcions reals integrables, la **convolució** de les funcions f i g, denotada per $f * g$ es defineix com

$$(f*g)(x) := \int_{-\infty}^{\infty} f(t)g(x-t)dt$$

Si les funcions f i g són acotades, i es compleix $\int_{-\infty}^{\infty} f(t)dt < \infty$, o bé $\int_{-\infty}^{\infty} g(t)dt < \infty$, aleshores la funció $(f * g)(x)$ està ben definida.

Aquesta operació és útil si es vol reconstruir un funció contínua, a partir d'una mostra discreta, tal com s'explica a continuació:

Definició 1.16. Sigui $f_{input}(x)$ una funció real acotada, que entendrem com la funció "original", i sigui $c(x)$ una funció de mostreig discret regular, expressada com:

$$c(x) = \sum_{n=-\infty}^{\infty} \mathbb{1}_{\{\delta n+\mu\}}(x) \quad \delta, \mu \in \mathbb{R},$$

on

$$\mathbb{1}_{\{\delta n+\mu\}}(x) = \begin{cases} 1 & \text{si } x = \delta n + \mu \\ 0 & \text{en cas contrari} \end{cases}.$$

Aleshores, es defineix la **mostra discreta** de $f_{input}(x)$ a partir de $c(x)$, com la funció

$$\hat{f}_{sampled}(x) = f_{input}(x) \cdot c(x).$$

Aquest procés de mostreig es pot veure a la Figura 1.7.

Definició 1.17. Reconstrucció. Sigui una funció discreta $\hat{f}_{sampled}(x)$, es pot obtenir una funció $f_{reconstructed}(x)$ que s'aproximi a la funció original $f_{input}(x)$, a partir d'una convolució

$$f_{reconstructed}(x) = \hat{f}_{sampled}(x) * h(x)$$

La funció $h(x)$ s'anomena ***kernel*** de convolució i ha de complir $\int_{-\infty}^{\infty} h(t)dt < \infty$. A la Figura 1.8 mostra gràficament la convolució d'un funció discreta.

1.4.2 Convolució de núvols de punts

Sigui $P = (p_i)n$ un núvol de punts. Es pot expressar el núvol de punts P com a un mapa discret $F\colon \mathbb{R}^3 \to \mathbb{R}$ tal que

$$F(x, y, z) = \sum_{i=1}^{n} \mathbb{1}_{p_i}(x, y, z)$$

on

$$\mathbb{1}_p(x, y, z) = \begin{cases} 1 & \text{si } (x, y, z) = p \\ 0 & \text{en cas contrari} \end{cases}.$$

Aleshores, si $h\colon \mathbb{R}^3 \to \mathbb{R}$ és un *kernel* de convolució, definim la convolució del núvol de punts P per el kernel h, com la funció $(F * h)$, que es pot expressar com:

$$(F * h)(x, y, z) = \sum_{i=1}^{n} h((x, y, z) - p_i)$$

En cas que el núvol de punts P segueixi un mostreig irregular, aleshores, per a cada punt p_i, es pot determinar un kernel h_i diferent, de forma que

$$(F * h)(x, y, z) = \sum_{i=1}^{n} h_i((x, y, z) - p_i)$$

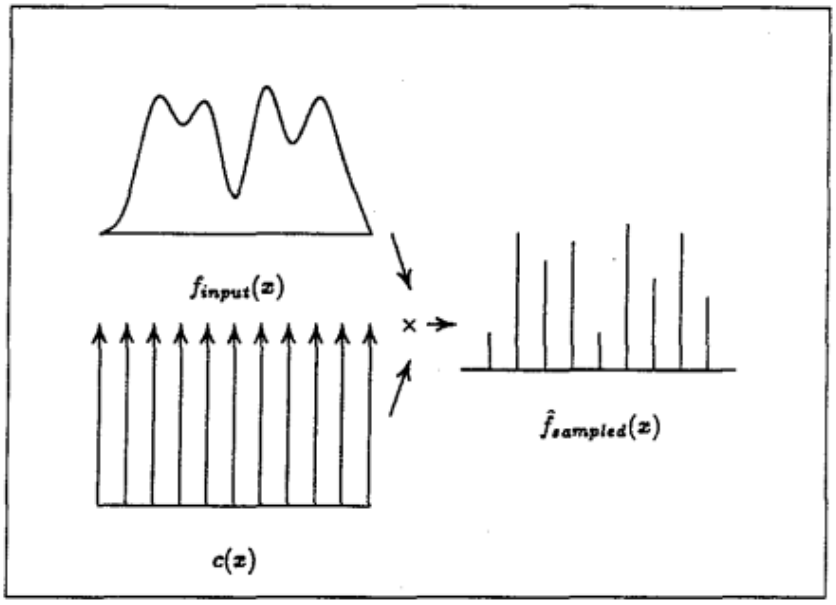


Figura 1.7: Mostreig discret d'una funció real continua [23]

1.5 Colors i il·luminació

Per tal de visualitzar una escena amb un cert "foto-realisme", cal que els objectes reaccionin a la llum. Per això és necessari definir el model i l'aproximació del transport de la llum, tal i com es detalla a continuació.

1.5.1 El color RGB

El **color** [16] és una propietat perceptiva causada per la llum quan aquesta interacciona amb l'ull. A causa de l'estructura anatòmica de l'ull, tots els colors visibles es poden obtenir a partir de la suma de fonts de llum de colors primaris. Aquests colors primaris són el **vermell**, **verd** i **blau**. Això ens permet representar qualsevol color com a una combinació lineal d'aquests 3 colors bàsics, d'aquí surt el que es coneix com l'espai de colors *RGB* [17].

Definició 1.18. Espai de colors *RGB*. L'espai de colors $\mathbb{RGB}$ representa un color com a un vector $c = (r, g, b) \in [0, 1]^3$, on r és la intensitat del **vermell**, g és la intensitat de **verd** i b és la intensitat de **blau**. El color visible d'un vector $c = (r, g, b)$ s'obté sumant les tres fonts lluminoses dels colors vermell, verd i blau, cada una amb les intensitats determinades per r, g, b respectivament. La Figura 1.9 mostra l'espai $\mathbb{RGB}$ com a un cub de colors.

Els colors *RGB* es poden sumar, restar i multiplicar component a component, truncant els valors a l'interval [0,1]:

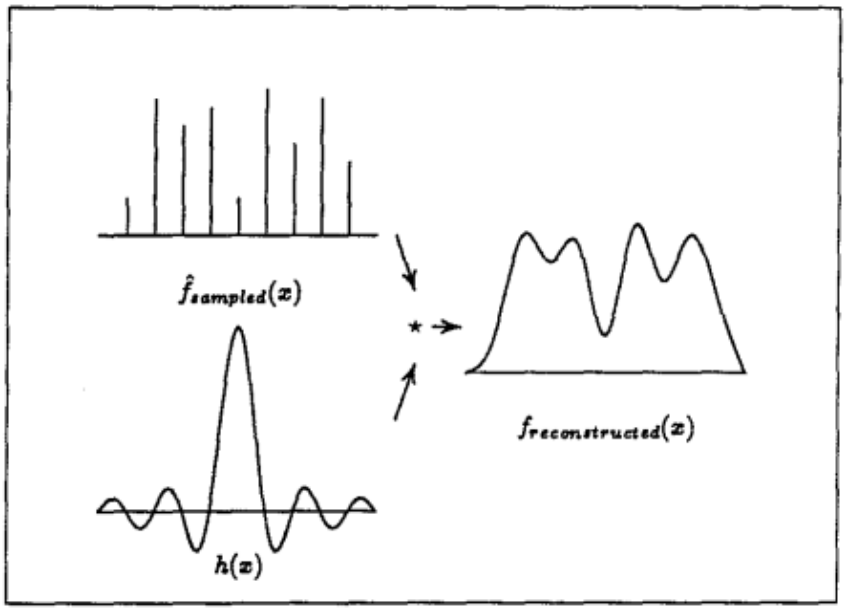


Figura 1.8: Convolució d'una mostra discreta [23]

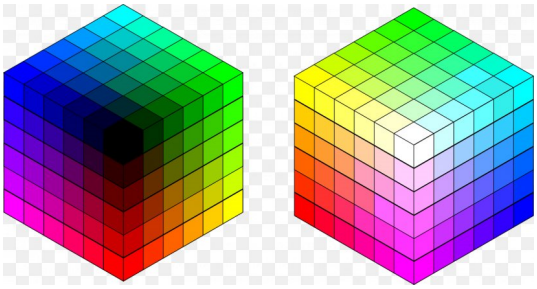


Figura 1.9: Representació visual de l'espai RGB [1]

Definició 1.19. Operacions de colors RGB. Sigui $c_1, c_2 \in [0,1]^3$ dos colors $\mathbb{RGB}$ tal que $c_1 = (r_1, g_1, b_1)$, $c_2 = (r_2, g_2, b_2)$ i $\lambda \in \mathbb{R}$, $\lambda \geq 0$, definim les següents operacions aritmètiques:

$$\begin{aligned}
\lambda \cdot c_1 &:= (\min(\lambda \cdot r_1, 1),\ \min(\lambda \cdot g_1, 1),\ \min(\lambda \cdot b_1, 1)) \\
c_1 + c_2 &:= (\min(r_1 + r_2, 1),\ \min(g_1 + g_2, 1),\ \min(b_1 + b_2, 1)) \\
c_1 - c_2 &:= (\max(r_1 - r_2, 0),\ \max(g_1 - g_2, 0),\ \max(b_1 - b_2, 0)) \\
c_1 \cdot c_2 &:= (r_1 r_2,\ g_1 g_2,\ b_1 b_2)
\end{aligned}$$

Per defecte, quan parlem d'un espai de colors ens estarem referint al format $\mathbb{RGB}$. Tot i això, per garantir la màxima generalitat, quan es defineixin objectes de visualització on el color concret és irrellevant, s'utilitzarà la notació d'un espai de colors $\mathcal{C}$ qualsevol.

1.5.2 Textures i imatges

Definició 1.20. Textura d'un model 3D. Sigui $M \subset \mathbb{R}^3$ un conjunt que representa un model 3D de visualització (M tan pot ser una superfície regular, una malla de triangles, o un núvol de punts), una **textura** de color $\mathcal{C}$ del model M és un mapa de la forma

$$T\colon M \to \mathcal{C}.$$

Definició 1.21. Imatge digital. Sigui $\mathcal{C}$ un espai de colors i sigui $m, n \in \mathbb{N}$, una **imatge digital** de **resolució** $n \times m$ és una matriu 2-dimensional de m files i n columnes de $\mathcal{C}$. Cada element de la imatge s'anomena **pixel**. (vegeu Figura 1.10)

Figura 1.10: Exemple d'imatge digital de 21×31 pixels [2]

1.5.3 Il·luminació

En gràfics d'ordinador, la il·luminació és el problema de simular el transport de la llum, que es genera en fonts d'il·luminació, fins que aquesta és captada per l'observador de la visualització.

Generalment, una tècnica d'il·luminació determina la intensitat, color i direcció dels rajos de llum que rep cada punt visible. Llavors, a partir del "material" del punt, es determina de quina forma la llum s'absorbeix, reflecteix o refracta, i quina llum arriba a l'observador.

El procés de càlcul d'il·luminació pot ser molt costós en temps de computació, sobretot si es busca el *foto-realisme*. Per l'objectiu del treball es busca una il·luminació que sigui

ràpida de calcular. Per aquest motiu, es considerarà una il·luminació de caràcter local, que es definirà de la següent manera:

Definició 1.22. Il·luminació local Un mètode d'**il·luminació** (o **ombrejat**) local és aquell que s'aplica el càlcul d'il·luminació per a cada punt de forma individual, és a dir, considerant que el model 3D està format únicament per aquell punt.

Aquest tipus d'il·luminació és ràpida de calcular, ja que no requereix traçades complexes de rajos de llum.

1.6 Càmeres

Un altre element important en la visualització, és el punt de vista de visualització i les parts visibles de l'escena. Aquests aspectes es defineixen amb l'objecte de *Càmera*.

1.6.1 Definició

Definició 1.23. Extensió projectiva de $\mathbb{R}^3$. L'extensió projectiva de l'espai afí $\mathbb{R}^3$, denotat per $\widetilde{\mathbb{R}^3}$ és un espai projectiu que presenta una bijecció

$$I\colon \widetilde{\mathbb{R}^3} \longleftrightarrow \mathbb{R}^3 \cup \mathbb{P}(\mathbb{R}^3) \tag{1.1}$$

Els punts $p = (x, y, z) \in \mathbb{R}^3$ es representen en l'espai $\widetilde{\mathbb{R}^3}$ per els punts $\widetilde{p} = [x : y : z : 1]$. Els punts $q = [x : y : z] \in \mathbb{P}(\mathbb{R}^3)$ es representen el l'espai $\widetilde{\mathbb{R}^3}$ per els punts $\widetilde{q} = [x : y : z : 0]$. Aquest punts $\widetilde{q}$ s'anomenem els punts de l'infinit.

Definició 1.24. Bases ortonormals Sigui $e_1, e_2, e_3 \in \mathbb{R}^3$ tres vectors, $\Delta = \{e_1, e_2, e_3\}$ és diu que es una **base ortonormal** de l'espai vectorial $\mathbb{R}^3$ sii

$$\|e_i\| = 1 \text{ per } i = 1, 2, 3$$
$$\langle e_i, e_j \rangle = 0 \text{ per } i \neq j$$

on $\langle , \rangle$ denota el producte escalar.

Una base ortonormal Δ genera un sistema de coordenades en $\mathbb{R}^3$ que manté l'estructura del producte escalar. És a dir, si $u, v \in \mathbb{R}^3$ són dos vectors amb coordenades $u = (u_x, u_y, u_z)_\Delta$, $v = (v_x, v_y, v_z)_\Delta$ en la base Δ, aleshores el vector $\langle u, v \rangle$ s'expressa com

$$\langle u, v \rangle = (u_x v_x, u_y v_y, u_z v_z)_\Delta$$

en coordenades en la base Δ

- Si $e_3 = e_1 \times e_2$, on $\times$ denota el producte vectorial, aleshores es diu que Δ genera un sistema de coordenades de **mà dreta**.
- Si $e_3 = -(e_1 \times e_2)$, aleshores es diu que Δ genera un sistema de coordenades de **mà esquerra**.

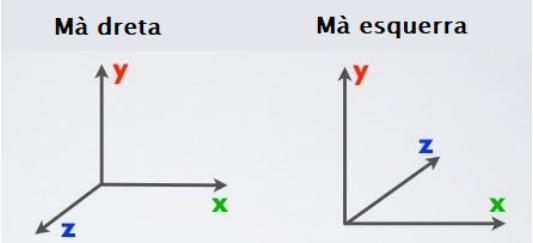


Figura 1.11: Sistemes de coordenades de mà dreta i mà esquerra

A la Figura 1.11 es mostra els dos tipus d'orientacions de coordenades.

Definició 1.25. Referència afí. Sigui $e_1, e_2, e_3 \in \mathbb{R}^3$ una base de l'espai vectorial $\mathbb{R}^3$, i sigui $o \in \mathbb{R}^3$. Es denota per $\Delta = \{o;\ e_1, e_2, e_3\}$ com a una **referencia afí** de $\mathbb{R}^3$, que genera un sistema de coordenades de forma que si $u = (x, y, z)_\Delta$ en coordenades en la base Δ, aleshores

$$u = xe_1 + ye_2 + ze_3 + o$$

Definició 1.26. Transformació de coordenades. Sigui Δ una referència de l'espai afí $\mathbb{R}^3$, es denota per $T_\Delta \colon \mathbb{R}^3 \to \mathbb{R}^3$ el mapa de transformació de coordenades canòniques a coordenades en la referència Δ.

Definició 1.27. Transformació de projecció. Sigui $P \colon \widetilde{\mathbb{R}^3} \to \widetilde{\mathbb{R}^3}$ una projecció en l'extensió projectiva de $\mathbb{R}^3$, es denota per $T_P \colon \mathbb{R}^3 \cup \mathbb{P}(\mathbb{R}^3) \to \mathbb{R}^3 \cup \mathbb{P}(\mathbb{R}^3)$, la transformació corresponent a la projecció P definida com

$$T_P = (I \circ P \circ I^{-1})$$

Definició 1.28. Càmera. Sigui Δ una **referència ortonormal** de l'espai afí $\mathbb{R}^3$ amb orientació de **mà esquerra** i T_Δ la transformació de coordenades. Sigui T_P la transformació corresponent a una projecció $P \colon \widetilde{\mathbb{R}^3} \to \widetilde{\mathbb{R}^3}$. Aleshores el parell (T_P, T_Δ) defineix una **càmera** de visualització, si es compleix que

$$T_P^{-1}([-1,1]^3) \subseteq \mathbb{R}^3 \tag{1.2}$$

Per convenció, $\Delta = \{O; v_1, v_2, v_3\}$, aleshores O denota la posició de la càmera, v_1 denota la direcció cap a la "dreta", v_2 denota la direcció cap a "dalt", v_3 denota la direcció cap "endavant".

Definició 1.29. Coordenades. Sigui una càmera (T_P, T_Δ), i sigui un punt $p \in \mathbb{R}^3$.

- S'anomenen **coordenades de món** (o **coordenades globals**) a les coordenades canòniques d'un punt de $\mathbb{R}^3$.
- S'anomenem **coordenades de vista** a les coordenades d'un punt p en la referència Δ. Es poden obtenir aplicant $T_\Delta(p)$. Les coordenades de vista són coordenades de **mà esquerra**.

- Anomenem **coordenades de càmera normalitzades** del punt p, a les coordenades del punt $(T_P \circ T_\Delta)(p)$, suposant que $(T_P \circ T_\Delta)(p) \in \mathbb{R}^3$. Per convenció, les coordenades de càmera normalitzades són un sistema de **mà dreta**, on la primera component denota la posició **horitzontal**, on més positiu significa més cap a la dreta; la segona component denota la posició **vertical**, on més positiu significa més cap amunt; i la tercera component denota la posició de profunditat, on més positiu significa més aprop.

  Les coordenades de càmera normalitzades serveixen per determinar la visibilitat dels punts. Sigui $p \in \mathbb{R}^3$, p és visible si, i només si,

  $$(T_P \circ T_\Delta)(p) \in [-1,1]^3.$$

  Anomenem al conjunt de tots els punts visibles

  $$U = (T_\Delta^{-1} \circ T_P^{-1})([-1,1]^3)$$

  el **tronc de visió** de la càmera (de l'anglès **frustum**).

En la Figura 1.12 es mostra el diagrama de transformacions de coordenades de forma resumida.

$$\textbf{coord. de món} \xrightarrow{T_\Delta} \textbf{coord. de vista} \xrightarrow{T_P} \textbf{coord. de càmera norm.}$$

$$\textbf{coord. de món} \xleftarrow[T_\Delta^{-1}]{} \textbf{coord. de vista} \xleftarrow[T_P^{-1}]{} \textbf{coord. de càmera norm.}$$

Figura 1.12: Diagrama de transformacions de coordenades

Definició 1.30. Projecció a pantalla. Sigui (T_P, T_Δ) una càmera, i sigui $p \in (T_\Delta^{-1} \circ T_P^{-1})([-1,1]^3)$ un punt dins del tronc de visió. Aleshores, es defineix la projecció del punt p a la **pantalla de càmera** com

$$(x,y) = (\pi \circ T_P \circ T_\Delta)(p)$$

on $\pi\colon \mathbb{R}^3 \to \mathbb{R}^2$, tal que $\pi(x,y,z) = (x,y)$. La funció $(\pi \circ T_P)$ s'anomena la **projecció a pantalla** de la càmera. Les coordenades (x,y) s'anomenen les coordenades **normalitzades de pantalla**

A continuació es donen expressions matemàtiques per a les transformacions T_Δ i T_P més utilitzades per a les visualitzacions 3D.

1.6.2 Transformacions de vista

Sigui $\Delta = \{O; v_1, v_2, v_3\}$ (referència ortonormal), la transformació de vista T_Δ amb es pot expressar com:

$$T_\Delta(x,y,z) = (\langle v_1, (x,y,z) - O\rangle, \langle v_2, (x,y,z) - O\rangle, \langle v_3, (x,y,z) - O\rangle)$$

on l'operació $\langle \cdot, \cdot \rangle$ és el producte escalar. O en forma matricial:

$$\begin{pmatrix} v_{1,x} & v_{1,y} & v_{1,z} & -\langle v_1, O \rangle \\ v_{2,x} & v_{2,y} & v_{2,z} & -\langle v_2, O \rangle \\ v_{3,x} & v_{3,y} & v_{3,z} & -\langle v_3, O \rangle \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La inversa T_Δ^{-1} és el canvi de coordenades de la referència Δ a la referencia canònica. S'expressa com:

$$T_\Delta^{-1}(x, y, z) = x \cdot v_1 + y \cdot v_2 + z \cdot v_3 + O$$

o en forma matricial:

$$\begin{pmatrix} v_{1,x} & v_{2,x} & v_{3,x} & O_x \\ v_{1,y} & v_{2,y} & v_{3,y} & O_y \\ v_{1,z} & v_{2,z} & v_{3,z} & O_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

1.6.3 Transformacions de projecció

Les projeccions P que ens interessen per la **visualització 3D** són aquelles en què $\pi \circ T_P$ sigui una projecció de l'espai al pla de tipus **ortogràfic** (o **isomètric**) o de tipus **perspectiu**.

Projeccions ortogràfiques

Una projecció ortogràfica és una projecció de $\mathbb{R}^3$ a un pla en què totes les línies de projecció són ortogonals al pla (veure Figura 1.13). Aquesta projecció és útil quan es vol tenir una idea de les distàncies exactes entre punts sense tenir en compte la distància al pla de projecció o l'orientació de la càmera.

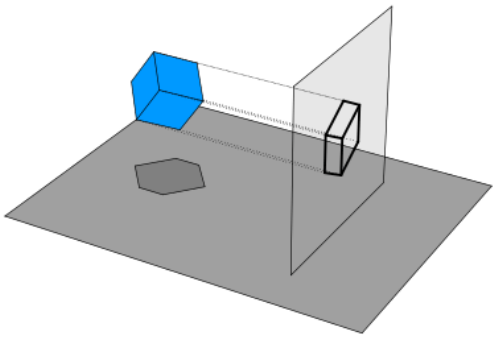


Figura 1.13: Projecció ortogràfica a un pla [3]

Notem que la projecció π és una projecció ortogràfica, i per tant, si desitgem que $\pi \circ T_P$ sigui ortogràfica, cal que la transformació T_P sigui lineal en l'espai afí. El més típic és

delimitar el tronc de visió a un cuboide rectangular ortogonal en els eixos de coordenades, i trobar la transformació lineal que envia el cuboide rectangular al cub unitari.

Un cuboide centrat a l'origen es pot determinar amb els valors $width, height, near, far$, de forma que $p_{min} = (-\frac{width}{2}, -\frac{height}{2}, near)$ i $p_{max} = (\frac{width}{2}, \frac{height}{2}, far)$ són els punts que delimiten del cuboide (veure Figura 1.14). Amb aquesta notació, la projecció $P\colon \widetilde{\mathbb{R}^3} \to \widetilde{\mathbb{R}^3}$ que envia el cuboide $(width, height, near, far)$ al cub unitari de forma lineal, es pot expressar en forma matricial com:

$$P = \begin{bmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

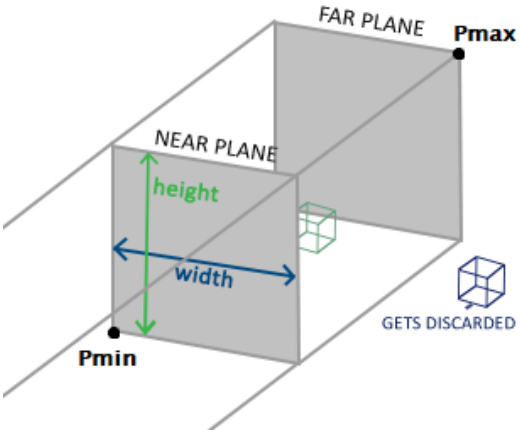


Figura 1.14: Tronc de visó ortogràfic [4]

Projeccions perspectives

Una projecció en perspectiva de $\mathbb{R}^3$ a un pla, és aquella en què les línies de projecció convergeixen a un mateix punt (veure figura 1.15). Aquesta projecció és la més natural visualment, ja que representa la forma de visió de l'ull humà i les càmeres fotogràfiques.

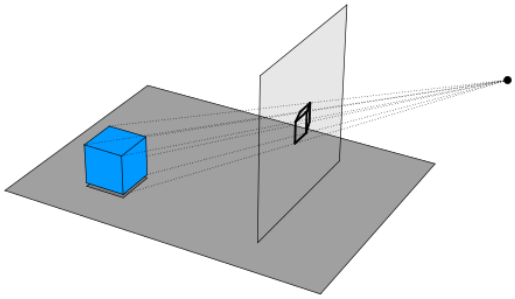


Figura 1.15: Projecció perspectiva a un pla [3]

El tronc de visió d'una transformació en perspectiva és una **piràmide** que convergeix a l'origen, i s'expandeix cap a l'eix de les z, **truncada** per dos plans diferents ortogonals a l'eix z (l'origen de coordenades no pot estar entremig dels dos). S'anomena *near plane*, al pla de menor distància a l'origen, i *far plane* a l'altre. L'amplada i alçada de la piràmide bé determinada per dos angles $l, v \in (0, \pi)$ respectivament. Els plans es poden determinar per les seves distàncies a l'origen $n, f \in \mathbb{R}$ $n < f$, on n és la distància del *near plane* i f la distància del *far plane*. L'amplada i alçada de la piràmide en el *near plane* es poden calcular com $width = 2n \cdot \tan \frac{l}{2}$ i $height = 2n \cdot \tan \frac{v}{2}$ respectivament (veure figura 1.16).

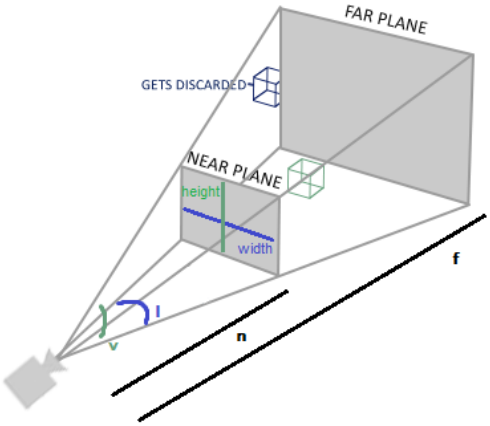


Figura 1.16: Tronc de visó perspectiu [4]

Donada una piràmide truncada (en la forma explicada anteriorment) determinada pels valors $width, height, f, n$, la projecció que envia la piràmide en el cub unitari $[-1, 1]^3$ es pot expressar en forma matricial com [8]:

$$P = \begin{bmatrix} \frac{n}{w} & 0 & 0 & 0 \\ 0 & \frac{n}{h} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2 \cdot f \cdot n}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

1.7 Definició del problema

Donat un **núvol de punts** (definició 1.9) $(p_i)_{i=1,\cdots,n}$ que s'ha extret d'una **superfície** (definició 1.5), es desitja un algorisme que generi **imatges digitals** (definició 1.21) en temps real [1], que simulin la visió d'un observador del núvol de punts a través d'una **càmera** (definició 1.28). A més, si la distribució de punts és prou bona, es vol que l'algorisme apliqui operacions de convolució (definició 1.4.2) per tal d'observar una reconstrucció aproximada de la superfície original, així com aplicar un model d'**il·luminació local** (definició 1.22). L'algorisme ha de poder reaccionar, en temps real, als canvis de la càmera, i canvis en la configuració de visualització.

[1] volen dir en un temps inferior als 30^{-1} segons

Capítol 2

Mètodes de visualització

2.1 Introducció

Com s'ha explicat al Capítol **??**, un procés de visualització 3D genera una imatge d'un model 3D des del punt de vista d'una càmera. Amb aquest objectiu, els algorismes de **visualització 3D** es divideixen típicament en 2 fases: la fase de càlcul de **visibilitat** i la fase de càlcul **d'il·luminació**.

- El **càlcul de visibilitat** aplica un mapa entre les coordenades discretes d'un píxel de la imatge (definició 1.21), i les coordenades normalitzades de pantalla (definició 1.30). Llavors, per a cada píxel de la imatge, s'ocupa de determinar quin punt (o punts) del model projecten en aquell píxel, i determina quins són els punts visibles i quins són els punts que queden amagats.
- El **càclul d'il·luminació** s'encarrega de determinar el color final de cada píxel de la imatge, utilitzant la informació geomètrica dels punts que s'han determinat en la fase de **visibilitat**. Les fases d'il·luminació normalment tenen en compte **textures de colors** del model (definició 1.20), i els rajos de llum que es reflecteixen en el punt del model.

En aquest capítol s'explicaran les dues metodologies principals pel càlcul de visibilitat: els mètodes de ***ray casting*** i els mètodes **projectius**.

Pel que fa al càlcul d'il·luminació, es presentarà el model *Phong*. Aquest model proporciona un càlcul d'il·luminació de superfícies de forma local (tal com s'ha especificat en la secció 1.22), i s'utilitza com a base d'il·luminació en la majoria de programes de visualització 3D interactius (sobretot videojocs, a causa de la seva rapidesa i eficàcia).

Per la qüestió de reconstrucció d'un núvol de punts (introduït en la secció 1.4.2), s'introduirà també un algorisme conegut com a *Splatting*. Aquest algorisme va ser dissenyat específicament per a visualitzar *voxels* (núvols de punts amb mostreig regular). Aquest mètode és d'interès, ja que utilitza un mètode **projectiu** pel càlcul de la visibilitat (el qual

permetrà una visualització interactiva, com es veurà), i aplica una operació de convolució per a reconstruir un model 3D continu. Finalment, a les conclusions d'aquest capítol, s'explicarà en quins mètodes està basat la nostra solució al problema i per a quins motius.

2.2 Càlcul de visibilitat

2.2.1 Transformació de píxels a pantalla de càmera

Definició 2.1. Transformacions de píxel a pantalla de càmera Donada una imatge digital de resolució $n \times m$ píxels (n columnes i m files), es transformen les coordenades d'un píxel, a coordenades normalitzades de pantalla amb les funcions:

$$\mathrm{l}\colon \mathbb{N}^2 \to \mathbb{R}^2 \tag{2.1}$$

$$\mathrm{l}(i,j) = \left(\frac{2i}{n} - 1 + \frac{1}{n}, \quad \frac{2j}{m} - 1 + \frac{1}{m}\right) \tag{2.2}$$

i

$$\mathrm{k}\colon \mathbb{R}^2 \to \mathbb{N}^2 \tag{2.3}$$

$$\mathrm{k}(x,y) = \left(\lfloor (x+1)\frac{n}{2})\rfloor, \quad \lfloor (y+1)\frac{m}{2})\rfloor\right) \tag{2.4}$$

$$\text{on } \lfloor\rfloor \text{ denota la funció } \textit{floor}(\textit{terra}), \tag{2.5}$$

La idea és dividir el quadrat unitari $[-1,1]^2$ en $n \times m$ rectangles de la mateixa mida, de forma que s'identifica cada píxel (i,j) de la imatge, amb el rectangle (i,j) del quadrat $[-1,1]^2$. Llavors la coordenada d'un píxel es correspon amb la coordena de del centre del seu rectangle. En la Figura 2.1 es mostra aquesta identificació per a una imatge de 8×8 píxels.

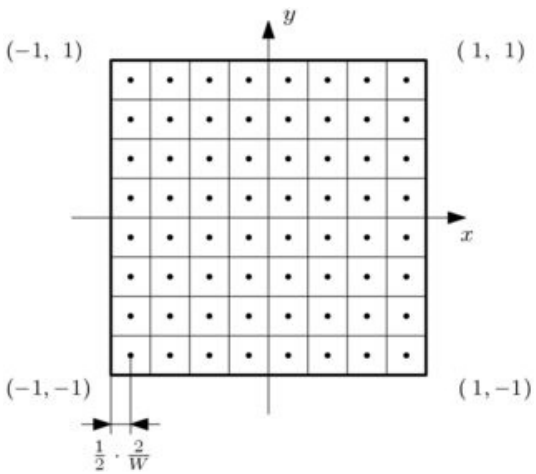


Figura 2.1: 8×8 píxels incrustats a l'interval $[-1,1]^2$

2.2.2 Mètode *Ray casting*

En el mètode de **ray casting** [20], per a cada píxel de la imatge es creen rectes, anomenades **rajos de visió**, i llavors es calcula el punt del model més proper a l'observador, que interseca en cada recta. Aquests punts són els que es consideren els punts visibles de cada píxel.

Sigui *Image* una imatge de resolució $n \times m$. Sigui (T_P, T_Δ) la càmera observadora amb $T := T_P \circ T_\Delta$. Sigui *Model* un model 3D (ja sigui una superfície, núvol de punts o malla de triangles), del qual es pugui calcular interseccions amb rectes [1]. Definim els punts visibles per a cada píxel com una matriu *VisiblePoints* de $\mathbb{R}^3$ amb la mateixa mida que la imatge *Image*. Aleshores l'algoritme de **ray casting** es pot expressar tal que es mostra en l'algoritme 1

Algorithm 1 Ray Casting (*Image*, T, *Model*)

1: $VisiblePoints :=$ *Matrix of* $\mathbb{R}^3$ *vectors*
2: **for all** pixel (i, j) **in** *Image* **do**
3: $(x, y) \leftarrow \mathrm{l}(i, j)$ ▷ (on l és la funció definida a 2.1)
4: $a \leftarrow T^{-1}(x, y, 1)$
5: $b \leftarrow T^{-1}(x, y, -1)$
6: $r \leftarrow a \vee b$ ▷ (raig de visió)
7: $U \leftarrow r \cap M$
8: **if** $U \neq \emptyset$ **then**
9: $VisiblePoints[i, j] \leftarrow$ **select** p **from** U **closest to** a **between** $[a, b]$
10: **return** *VisiblePoints*

A la Figura 2.2 es mostra un exemple d'un **raig de visió** r per als dos tipus de càmeres. A la Figura 2.3 es mostra un exemple d'una visualització completa amb el mètode de **ray casting**, on després de calcular el punt visible per a cada píxel, es determina el color del píxel a partir del color del punt del model.

2.2.3 Mètode projectiu

En un mètode **projectiu**[7], per tal de calcular els punts visibles que projecten en cada píxel, el que es fa, és aplicar directament la transformació de càmera $T = (T_P \circ T_\Delta)$ en el model 3D, per tal de convertir-lo en un model 2D en coordenades de pantalla normalitzades. Aleshores, el model 2D projectat s'ha de discretitzar per convertir-lo en un conjunt de píxels. Els punts que queden "superposats", en la mateixa recta de projecció d'un píxel, s'han de descartar per tal de quedar-se amb el punt més proper a càmera.

Aquest tipus de mètode se sol utilitzar quan el model 3D està compost per primitives geomètriques simples (com per exemple una malla de triangles, o un núvol de punts) i d'aquesta forma obtenir el model 2D projectat amb càlculs simples, per a cada primitiva.

[1] Aquí sobre el problema de la intersecció d'un núvol de punts amb rectes, que no és trivial de resoldre

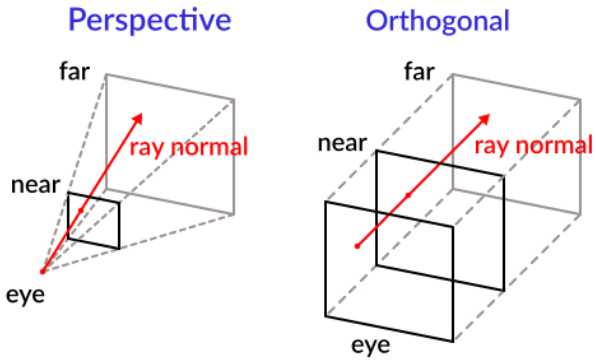


Figura 2.2: En vermell es mostren els raigs de visió per una càmera perspectiva i ortogràfica

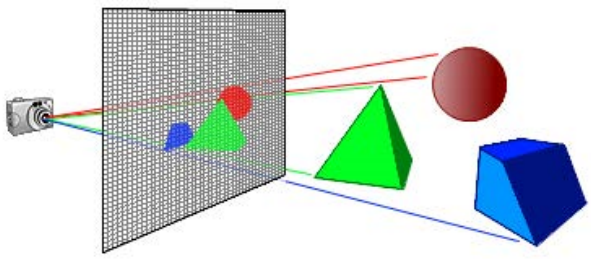


Figura 2.3: Exemple d'una visualització 3D amb ray casting

Aquí es mostrarà l'algorisme d'un mètode projectiu per a una malla de triangles, i de quina forma es pot modificar per obtenir un algorisme projectiu simple per a núvols de punts.

La projecció d'un triangle de l'espai a un pla, és senzilla, ja que només es requereix projectar els tres vèrtexs del triangle, i llavors unir els nous vèrtexs projectats per obtenir el triangle projectat. Així doncs, sigui *Model* una malla de triangles, *PuntsVisibles* una matriu $n \times m$ de $mathbbR^3$ on cada element correspon al punt visible d'un píxel, i $T = (T_P \circ T_\Delta)$ la transformació d'una càmera. Aleshores l'algoritme projectiu es pot expressar tal com es mostra en l'algorisme 2

Algorithm 2 Projective Method (*Image*, *T*, *Model*)

1: $VisiblePoints$:= *Matrix of* $\mathbb{R}^3$ *vectors*
2: **for all** Triangle (p_1, p_2, p_3) **in** *Model* **do**
3: $(g_1, g_2, g_3) \leftarrow (T(p_1), T(p_2), T(p_3))$
4: $(x_k, y_k) := \pi(g_k)$ **for** $k = 1, 2, 3$ ▷ (on $\pi(x, y, z) = (x, y)$)
5: $(i_k, j_k) \leftarrow \mathrm{k}(x_k, y_k)$ **for** $k = 1, 2, 3$ ▷ (on k és la funció definida a 2.3)
6: $i_{min} \leftarrow \min\{i_1, i_2, i_3\}$
7: $j_{min} \leftarrow \min\{j_1, j_2, j_3\}$
8: $i_{max} \leftarrow \max\{i_1, i_2, i_3\}$
9: $j_{max} \leftarrow \max\{j_1, j_2, j_3\}$
10: **for** $\hat{i}$ **in range** $i_{min}, \ldots, i_{max}$ **do**
11: **for** $\hat{j}$ **in range** $j_{min}, \ldots, j_{max}$ **do**
12: $center \leftarrow \mathrm{l}(\hat{i}, \hat{j})$ ▷ (on l és la funció definida a 2.1)
13: $z_k \leftarrow g_{k,z}$ **for** $k = 1, 2, 3$
14: **if** $center \in [-1, 1]^2$ $center$ **inside** Triangle $((x_1, y_1), (x_2, y_2), (x_3, y_3))$ **then**
15: $depth \leftarrow$ **interpolate** (z_1, z_2, z_3) **by** $center$
16: **if** $depth \in [-1, 1]$ **then**
17: **if** $VisiblePoints[\hat{i}, \hat{j}]$ **is empty then**
18: $VisiblePoints[\hat{i}, \hat{j}] \leftarrow (center_x, center_y, depth)$
19: **else if** $depth <$ **depth**$(VisiblePoints[\hat{i}, \hat{j}])$ **then**
20: $VisiblePoints[\hat{i}, \hat{j}] \leftarrow (center_x, center_y, depth)$
21: **return** $VisiblePoints$

En la línia 14 de l'algorisme 2, es determina si un punt cau dins d'un triangle; i en la línia 15, es calcula la profunditat d'un punt del triangle, interpolant les profunditats dels 3 vèrtexs. Aquests dos càlculs es poden obtenir mitjançant les coordenades baricèntriques del triangle. Sigui u_1, u_2, u_3 un triangle en $\mathbb{R}^2$, aleshores les coordenades baricèntriques (α, β, γ) d'un punt p es poden calcular amb les fórmules:

$$\alpha = \frac{area\{p, u_2, u_3\}}{area\{u_1, u_2, u_3\}}$$
$$\beta = \frac{area\{p, u_1, u_3\}}{area\{u_1, u_2, u_3\}}$$
$$\gamma = \frac{area\{p, u_1, u_2\}}{area\{u_1, u_2, u_3\}}$$

on $area\{a, b, c\}$ denota l'area del triangle generat per a, b, c

El punt p està dins del triangle u_1, u_2, u_3 sii $\alpha, \beta, \gamma \in [0, 1]$ i $\alpha + \beta + \gamma = 1$. Per evitar errors de precisió en la comprovació $\alpha + \beta + \gamma = 1$, el que es fa és calcular γ com $\gamma = 1 - \alpha - \beta$. D'aquesta forma el punt p cau dins del triangle sii

$$0 \leq \alpha \leq 1$$
$$0 \leq \beta \leq 1$$
$$0 \leq \gamma \leq 1$$

Si posem per z_1, z_2, z_3 les profunditats dels tres vèrtexs del triangle, aleshores es pot interpolar la profunditat d'un punt p dins del triangle amb coordenades baricèntriques (α, β, γ) de la següent manera:

$$z_p = \alpha z_1 + \beta z_2 + \gamma z_3$$

El procés de determinar si el centre d'un píxel cau dins d'un triangle (o d'una primitiva geomètrica 2D qualsevol), s'anomena **rasterització**. En la Figura 2.4 es representa visualment la rasterització d'un triangle.

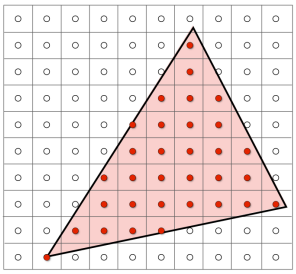


Figura 2.4: Exemple de rasterització d'un triangle. Els punts vermells denoten els píxels que cauen dins del triangle [5]

Núvol de punts

En cas que el model sigui un núvol de punts, es pot simplificar el algorisme 2, segons l'estratègia mostrada en l'algorisme 9.

Algorithm 3 Projective Method (*Image*, *T*, *PointCloud*, *R*)

VisiblePoints := *Matrix of* $\mathbb{R}^3$ *vectors*
for all p **in** *PointCloud* **do**
    $g \leftarrow T(p)$
    $(x, y) := \pi(g)$ ▷ (on $\pi(x, y, z) = (x, y)$)
    $(i, j) \leftarrow \mathrm{k}(x, y)$ ▷ (on k és la funció definida a 2.3)
    for $\hat{i}$ **in range** $i - R, \ldots, i + R$ **do** ▷ (on R és el radi del rectangle de píxels que genera cada punt)
        for $\hat{j}$ **in range** $j - R, \ldots, j + R$ **do**
            $center \leftarrow \mathrm{l}(\hat{i}, \hat{j})$ ▷ (on l és la funció definida a 2.1)
            $depth \leftarrow p_z$
            if $VisiblePoints[\hat{i}, \hat{j}]$ **is empty then**
                $VisiblePoints[\hat{i}, \hat{j}] \leftarrow (center_x, center_y, depth)$
            else if $depth <$ **depth**$(VisiblePoints[\hat{i}, \hat{j}])$ **then**
                $VisiblePoints[\hat{i}, \hat{j}] \leftarrow (center_x, center_y, depth)$
return *VisiblePoints*

2.3 Càlcul d'il·luminació: model *Phong*

Un cop determinat el punt visible del model 3D que projecta en un píxel, cal calcular el color del píxel de la imatge usant la informació del punt. D'això se n'ocupa la fase d'il·luminació.

Com s'ha explicat en la introducció del capítol 2.1, el mètode que s'utilitzarà pel càlcul d'il·luminació en la solució del treball, és el mètode **Phong** [15]. El model **Phong** descriu la forma en que una superfície reflecteix la llum localment, mitjançat una suma de tres colors $\mathbb{RGB}$ (veure secció 1.18): el color **ambient**, el color **difús** i el color **especular**. En la Figura 2.5 es mostren aquests tres termes per a una superfície acolorida. A continuació es detallen els càlculs dels colors per a cada terme, i la fórmula general del mètode Phong.

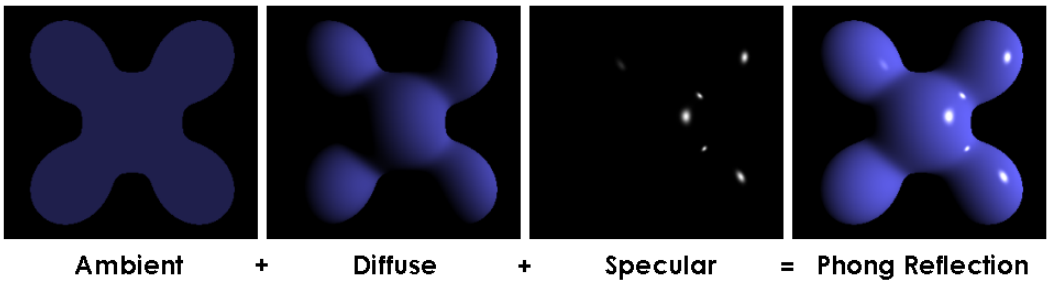


Figura 2.5: Il·lustració visual de la fórmula Phong [6]

Definició 2.2. Llum. Sigui S una superfície. En el model Phong, una **llum** es defineix com a una tupla (i_a, i_d, i_s, L) composta per tres camps de llum i un camp vectorial

$$i_a \colon S \to \mathbb{RGB}$$
$$i_d \colon S \to \mathbb{RGB}$$
$$i_s \colon S \to \mathbb{RGB}$$
$$L \colon S \to \mathcal{S}^2 \text{ (l'esfera de radi 1 centrada a l'origen)}$$

tal que, i_a, i_d, i_s representen les intensitats de color **ambient**, **difús**, **especular** respectivament, i L representa la direcció del raig de llum, per a qualsevol punt de la superfície S.

Definició 2.3. Material. Sigui S una superfície. En el model Phong, el **material** de S es defineix com a una tupla (k_a, k_d, k_s, α) de tres camps de colors, i un escalar

$$k_a \colon S \to \mathbb{RGB}$$
$$k_d \colon S \to \mathbb{RGB}$$
$$k_s \colon S \to \mathbb{RGB}$$
$$\alpha \colon \ \to \mathbb{R}$$

k_a, k_d, k_s s'anomenen els coeficients de reflexió del color **ambient**, **difús**, **especular** respectivament, i α s'anomena el coeficient de brillantor, per a cada punt de S.

Definició 2.4. Direcció de càmera. Sigui S una superfície, i sigui $T = T_P \circ T_\Delta$ la transformació d'una càmera. Donat un punt $p \in S$, denotem per $g := T^{-1}(x, y, 1)$ on $(x, y) = \pi(T(p))$, aleshores el raig de projecció de té direcció $v = \frac{g-p}{\|g-p\|}$. Definim per

$$V \colon S \to \mathcal{S}^2$$

com la funció que retorna la direcció $v = \frac{g-p}{\|g-p\|}$ per a qualsevol $p \in S$.

Definició 2.5. Vector de reflexió. Sigui S una superfície. Sigui $N \colon S \to \mathcal{S}^2$ un camp vectorial que representa el camp normal de la superfície, i sigui L el camp vectorial de la direcció d'un llum. Es defineix la reflexió de L a través de l'eix N en un punt p com:

$$R \colon S \to \mathcal{S}^2$$
$$R(p) = 2\langle L(p), N(p)\rangle N(p) - L(p)$$

En la Figura 2.6 es pot veure el vector $R(p)$ en un punt de la superfície.

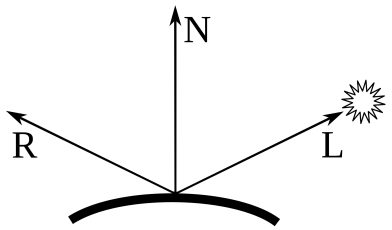


Figura 2.6: Reflexió del vector de llum amb l'eix de la normal

Definició 2.6. Termes ambient, difús i especular Sigui S una superfície i sigui $N \colon S \to \mathcal{S}^2$ un camp vectorial que representa el camp normal de la superfície (N no cal que sigui diferenciable ni contínua). Sigui $V(p)$ el vector de direcció de la càmera. Sigui una llum (i_a, i_d, i_s, L), i sigui (k_a, k_d, k_s, α) el material de S. Aleshores per a un punt $p \in S$, es defineix:

- El terme **ambient** com:
$$c_a(p) := k_a(p) i_a(p)$$
- El terme **difús** com:
$$c_d(p) := k_d(p) i_d(p) \langle L(p), N(p)\rangle$$
- El terme **especular** com:
$$c_s(p) := k_s(p) i_s(p) \langle R(p), V(p)\rangle^\alpha$$

Definició 2.7. Fórmula de phong. Amb les mateixes hipòtesis que la definició 2.6, per a una llista de M llums $(i_a, i_d, i_s, L)_m$ per $m = 1, \ldots, M$, la fórmula de phong determina el

color $C(p)$ per a un punt $p \in S$ com:

$$C(p) = \sum_{m=1}^{M} c_{a,m}(p) + c_{d,m}(p) + c_{s,m}(p)$$
$$= \sum_{m=1}^{M} k_a(p) i_{a,m}(p) + k_d(p) i_{d,m}(p) \langle L_m(p), N(p) \rangle + k_s(p) i_{s,m}(p) \langle R_m(p), V(p) \rangle^{\alpha}$$

2.4 Splatting

El mètode *Splatting* va ser proposat l'any 1991 per Lee Alan Westover [23], com a un algoritme de visualització de **vòxels** (núvols de punts amb mostreig regular). Utilitza un mètode **projectiu** avançat, per tal de visualitzar una reconstrucció d'un model continu a partir del núvol de punts, aplicant una operació de **convolució** 1.4.2.

Abans d'entendre com funciona l'algorisme de *Splatting*. Cal introduir el concepte de **volums semitransparents** i **dispersió** d'un raig de llum.

2.4.1 Definicions

Definició 2.8. Volum semitransparent. S'anomena un **volum semitransparent** a $\mathbb{R}^3$ com una funció

$$F \colon \mathbb{R}^3 \to [0,1]$$
$$F(x,y,z) = a$$

a és la "opacitat", on $a = 0$ vol dir totalment transparent, i $a = 1$ vol dir totalment opac.

Definició 2.9. Dispersió d'un raig de llum. Sigui F un volum, i sigui $p_0, p_1 \in \mathbb{R}^3$ dos punts, que denoten un segment. Aleshores, es modela la pèrdua d'energia d'un raig, que s'origina a p_0 i acaba a p_1, a través del volum F, amb l'expressió

$$I(p_o, p_1) = \begin{cases} \exp\left\{ \int_{p_0}^{p_1} \log\left(1 - F(S)\right) dS \right\} & \text{si } (F(S) \neq 1 \quad \forall t \in [0,1] \\ 0 & \text{en cas contrari} \end{cases}$$

on S parametritza la recta que passa per p_0 i p_1.

I denota el coeficient de pèrdua d'energia, és a dir, si es té un raig de llum amb energia inicial I_{inici}, i recorre el segment entre p_0, p_1, aleshores l'energia final serà

$$I_{fi} = I_{inici} \cdot I(p_0, p_1)$$

Intuïtivament, la fórmula de I es pot entendre com una "multiplicació contínua" de la transparència del volum. Per exemple, si tenim un volum amb opacitat constant a_0, aleshores, i el segment d'un raig té una longitud de s, aleshores el coeficient I de pèrdua d'energia serà

$$I = (1 - a_0)^s$$

Definició 2.10. Color d'un volum. Sigui $F\colon \mathbb{R}^3 \to [0,1]$ un volum semitransparent, es denota el color del volum amb una funció

$$C\colon \mathbb{R}^3 \to \mathbb{RGB}$$
$$C(x,y,z) = (rgb)$$

Definició 2.11. Model d'il·luminació per un volum semitransparent. Sigui un volum semitransparent amb funció $F\colon \mathbb{R}^3 \to [0,1]$ i color $C\colon \mathbb{R}^3 \to \mathbb{RGB}$. Sigui un raig de **visió** d'una càmera, que va del punt p_0 al punt p_1. Aleshores, utilitzant la funció de dispersió I (2.9), es pot calcular el color del volum F, C, a través del raig $[p_0, p_1]$ amb l'expressió

$$color(p_0,p_1) = \int_{p_0}^{p_1} C(S)\cdot F(S)\cdot I(p_0,S)\, dS$$

Per l'algorisme de visualització, interessa tenir una versió del model discreta. L'expressió d'il·luminació anterior es pot discretitzar de la següent manera:

Definició 2.12. Model d'il·luminació per una mostra discreta d'un volum semitransparent. Suposant que un raig de visió ha mostrejat, de forma ordenada, els colors C_i i opacitats F_i per $i = 1,\dots,n$ a través d'un volum semitransparent. L'expressió 2.11 per a la mostra discreta és:

$$color = \sum_{i=1}^{n} C_i \cdot F_i \cdot \prod_{j=1}^{i-1} (F_j - 1)$$

Definició 2.13. Efecte i petjada d'un kernel Sigui $(p_i)_{i=1,\dots,n}$ un núvol de punts, i $(c_i)_{i=1,\dots,n}$, $c_i \in \mathbb{RGB}$ un color corresponent a cada punt. Aleshores, com s'ha definit a la secció 1.4.2, es poden obtenir les funcions

$$F(x,y,z) = \sum_{i=1}^{n} h((x,y,z) - p_i)$$
$$C(x,y,z) = \sum_{i=1}^{n} c_i \cdot h((x,y,z) - p_i)$$

on $h\colon \mathbb{R}^3 \to \mathbb{R}$ és un **kernel** de convolució adient.

F i C, tal com s'ha definit en la secció 2.4.1, formen una reconstrucció d'un volum semitransparent, on F és la opacitat i C és el color.

L'algorisme de **splatting** ofereix un mètode per visualitzar el volum semitransparent determinat per F i C de forma eficient. Ho fa mitjançant una aproximació de l'**efecte** que genera el kernel h en cada punt. L'**efecte** es defineix de la següent manera:

Sigui $h\colon \mathbb{R}^3 \to \mathbb{R}$ un kernel de convolució. Es defineix l'**efecte** de h en un punt $p \in \mathbb{R}^3$, com la funció:

$$effect_p\colon \mathbb{R}^3 \to \mathbb{R}$$
$$effect_p(x,y,z) = h((x,y,z) - p)$$

Sigui una càmera amb transformació T, es pot "projectar" el volum generat per la funció **efecte** amb la transformació T, per tal d'obtenir una expressió de l'efecte en coordenades de pantalla. Aquesta projecció s'aconsegueix integrant la funció al llarg d'un raig de visió i s'interpreta com a una aproximació de la formula 2.11. La funció resultant s'anomena **footprint**, definida com:

$$\begin{aligned} footprint_p &: [-1,1]^2 \to \mathbb{R} \\ footprint_p(x,y) &= \int_{-1}^{1} effect_p(T^{-1}(x,y,z))\,dz \end{aligned} \tag{2.6}$$

Definició 2.14. Kernel Gausià. Un dels kernels de convolució més utilitzats en tot tipus de contextos, degut a que produeix una reconstrucció suau i dona un aspecte orgànic és la funció **gaussiana**:

$$h_\lambda(x,y,z) = \exp\left\{-\frac{\|(x,y,z)\|^2}{2\lambda^2}\right\}, \quad \lambda \in \mathbb{R}$$

2.4.2 Mètode

Per a cada punt p del núvol, el mètode de *Splatting* primer determina la funció $footprint_p(x,y)$ a partir del *kernel* de convolució *gausià* (def 2.14). Aleshores, de igual manera que un mètode projectiu per a núvols de punts, es projecta cada punt p en el seu píxel (i,j) corresponent de la pantalla, i genera un quadrat de píxels centrat en (i,j) d'una mida R. Llavors, a cada píxel del quadrat se li assigna el valor de $footprint_p$ utilitzant les coordenades del centre del píxel.

Com que cada punt genera petjada de múltiples píxels, es esperable que, diverses petjades $footprint_p$ afectin a un mateix píxel. Suposant que per a un píxel concret, hi ha petjades diferents que l'afecten amb colors diferents, el que proposa el mètode és barrejar els diferents colors utilitzant la fórmula de la def. 2.12. Com que aquesta fórmula requereix que la mostra estigui ordenada al llarg del raig de visió (l'eix z en coordenades de càmera), el que es fa és ordenar els punts del núvol en funció de la distància a càmera de major a menor distància.

En l'algoritme 4 es mostra el mètode de *Splatting*.

Algorithm 4 Splatting (*Image*, T, *PointCloud*, h, R)

sort points **in** *PointCloud* **by** distance to camera (from back to front)
for all $p, color$ **in** *PointCloud* **do** ▷ (on p és el punt i c el color)
    $footprint_p(x,y) := \int_{-1}^{1} h(T^{-1}(x,y,z) - p)\, dz$
    $g \leftarrow T(p)$
    $(x,y) := \pi(g)$ ▷ (on $\pi(x,y,z) = (x,y)$)
    $(i,j) \leftarrow \mathrm{k}(x,y)$ ▷ (on k és la funció definida a 2.3)
    for $\hat{i}$ **in range** $i - R, \ldots, i + R$ **do** ▷ (R és el radi del quadrat de píxels fixat per a tots els punts.)
        for $\hat{j}$ **in range** $j - R, \ldots, j + R$ **do**
            $coords \leftarrow \mathrm{l}(\hat{i}, \hat{j})$ ▷ (on l és la funció definida a 2.1)
            $a \leftarrow footprint_g(coords)$
            $Image[\hat{i}, \hat{j}] \leftarrow color \cdot a + (1 - a) \cdot Image[\hat{i}, \hat{j}]$
return *Image*

La clau de l'algorisme està en tenir un calcul senzill de $footprint_p$. El cas més optimista, és que la projecció del kernel h a càmera tingui la mateixa forma per a tots els punts, i només variï per una translació. Això permetria que només s'hagués de calcular la funció $footprint$ un cop per a cada imatge de visualització. Això, però, només passa en pocs casos. Un d'ells és que la càmera sigui de tipus ortogràfica i que el kernel sigui totalment simètric (com la funció Gausiana).

2.5 Conclusions

L'algorisme *splatting* ens ofereix el tipus de visualització de núvol de punts que busquem, però caldran algunes modificacions per adaptar-lo al cas de núvols de punts de superfícies amb un mostreig aleatori. El capítol 3 s'explica com s'ha aplicat el mètode per el nostre cas i les modificacions que s'han introduït.

Capítol 3

Solució teòrica

3.1 Introducció

Com hem dit a la secció **??**, la solució del treball està basada en el mètode *Splatting* (2.4) per oferir una visualització d'un núvol de punts amb una reconstrucció contínua, i incorpora el mètode d'il·luminació *Phong* (2.3).

El model és un núvol de punts format per parells (p_i, c_i) per $i = 1, \ldots, N$ que representa una mostra finita d'una superfície amb una textura de colors (p_i és la posició i c_i el color).

El mostreig del núvol se suposarà de tipus aleatori. Per aquest motiu seria convenient aplicar funcions de filtratge i normalització abans de la visualització. A més, com es desitja aplicar el model de *phong* es necessita determinar un vector normal en cada punt, per tant ens cal un mètode per aproximar els plants tangents de la superfície original en cada punt p_i.

Això porta a dividir l'algorisme en dues parts, una part per processar la mostra de punts (filtrar, normalitzar i calcular vectors normals), i l'altre de visualitzar el model processat.

3.2 Algorisme

L'algoritme està dividint en dos parts anomenades: *Processament de dades* (3.3) i *Visualització* (3.4).

- La part de *Processament de dades* serveix per adaptar millor les dades del núvol de punts de cara a una millor visualització, així com calcular la informació requerida que no està present de forma inicial. L'algorisme limitarà la "sessió" [1] de visualitza-

[1] "sessió" referint-se a una execució del programa

ció, a un únic núvol de punts constant. Això permetrà que la part de *Processament de dades* només s'hagi d'executar un cop al començament de la sessió.

- La part de *Visualització* és el procés que s'encarrega de generar les imatges a partir del núvol de punts i la càmera observadora. La *visualització* s'executa en bucle fins que l'usuari decideix parar el programa.

En la Figura 3.1 es mostra l'execució d'un programa dividit en les dues parts.

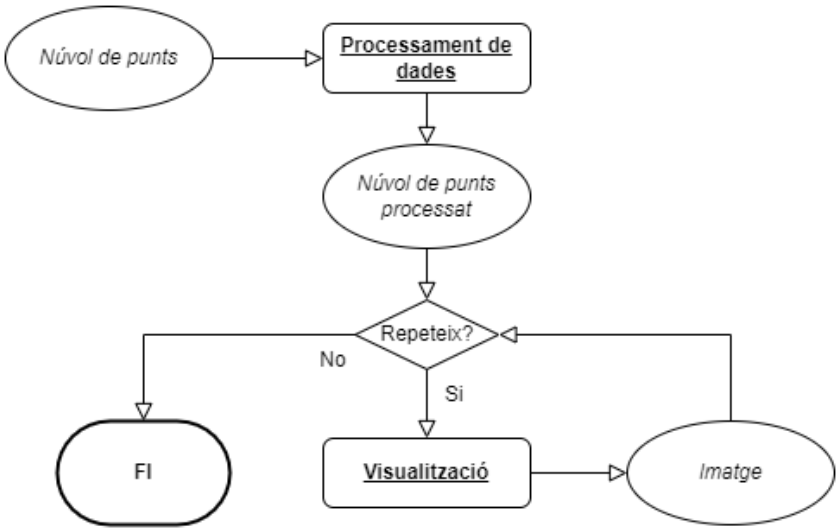


Figura 3.1: Flux general del programa

A continuació es detalla cada part del algorisme.

3.3 Processament de dades

El processament de dades està compost per una sèrie de passos seqüencials: **filtratge**, **estandardització**, **càlcul de radis d'entorn** i **càlcul de normals**, com es mostra a la figura 3.2.

Figura 3.2: Flux del processament de dades

En tot el processament de dades s'utilitzaran les funcions de **distància veïna**, **veïns pròxims**, i **radi d'entorn** definides en 1.12, 1.13 i 1.14.

A continuació es detalla cada pas.

3.3.1 Filtratge

La idea del filtratge és eliminar possibles punts de la mostra que corresponguin a errors de mesura o observacions atípiques.

Sigui $P = (p_i)_{i=1,\dots,n}$ els punts del núvol. Primer de tot, es suposa que si $i \neq j$ aleshores $p_i \neq p_j$. Si en la mostra original això no es compleix, aleshores s'hauran d'eliminar els punts repetits.

Per filtrar les observacions atípiques, es considera la distribució de la **distància veïna** $\mathrm{dv}(p)$, per a un $p \in P$, com a variable aleatòria. Com a exemple, a la figura 3.3, es pot veure la distribució de $\mathrm{dv}_P(p)$ per a una mostra finita de punts aleatoris del quadrat $[-1,1]^2$. S'observa que és una distribució **asimètrica positiva**. L'objectiu és filtrar els valors que siguin atípicament grans. S'ha d'anar amb cura perquè si es filtra massa liberalment, es perdrà fidelitat de visualització.

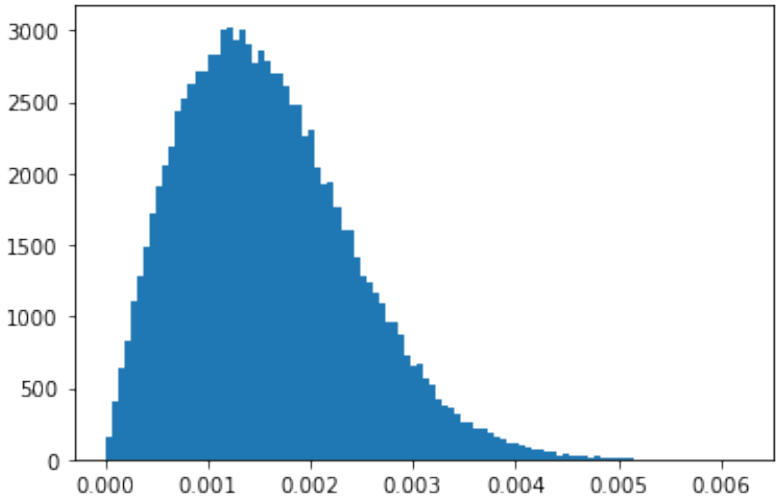


Figura 3.3: Distribució de $\mathrm{dv}(p)$ per a 10^5 punts aleatoris en $[-1,1]^2$

El mètode que s'utilitzarà per filtrar valors grans, és una modificació del *z-score* [11] tenint en compte només els valors superiors a la mitjana. L'algorisme és el següent:

Algorithm 5 Filtratge (points, threshold)

distances $\leftarrow$ **list** { $\mathrm{dv}(p)$ | **for** p **in** points }
mean_dist $\leftarrow$ **mean of** distances
r_distances $\leftarrow$ **list** { d | **if** $d \geq$ mean_dist **for** d **in** distances }
rN $\leftarrow$ **size of** r_distances
r_std_dev $\leftarrow \sqrt{\frac{1}{\mathrm{rN}} \sum_{i=1}^{rN} (\mathrm{r\_distances}[i] - \mathrm{mean\_dist})^2}$
for all p **in** points **do**
    if $\mathrm{dv}(p) >$ mean_dist **and** $\frac{\mathrm{dv}(p) - \mathrm{mean\_dist}}{\mathrm{r\_std\_dev}} >$ threshold **then**
        remove p **from** points
return points

threshold denota el valor llindar pel *z-score* modificat. El valor adient dependrà de cada mostra concreta, però un valor de 5 ens serveix prou bé en la majoria de casos.

3.3.2 Estandardització

Estandarditzar és el procés d'aplicar la homotècia més translació adient per assegurar que els punts quedin inclosos en l'esfera de centre $(0,0,0)$ i radi 1. Aquest procés és útil per garantir una visualització consistent amb una càmera inicial per a tot tipus de dades.

Algorithm 6 Estandaritació (points)

n $\leftarrow$ **size of** points
center $\leftarrow \frac{1}{n} \sum_{i=1}^{n} p_i$
radius $\leftarrow \max_{i=1,\dots,n} \|p_i - \mathrm{center}\|$
for all p **in** points **do**
    $p \leftarrow \frac{p - \mathrm{center}}{\mathrm{radius}}$
return points

3.3.3 Càlcul de radis d'entorn

En aquest pas, es busca calcular un $r_p \in \mathbb{R}$ per a tot punt $p \in P$ del núvol, de forma que, la bola $\bar{B}(p, r_p) \subset \mathbb{R}^3$, de centre p i radi r_p, determini l'entorn de "contacte" per el punt p. Posteriorment, aquesta bola $\bar{B}(p, r_p)$ s'utilitzarà com l'entorn per aproximar el vector normal de la superfície i com l'acotació de les funcions $efecte$ i $footprint$ de cada punt, en la part de visualització (veure definició 2.13).

Per calcular el valor de r_p, el que es fa és buscar la mitjana de la distància veïna per els k punts més propers a p. És a dir, sigui $P = (p_i)_{i=1,\dots,n}$, es denota per el k-èssim punt més proper a p, com:

$$\mathrm{last}\ \mathrm{V}(p,k)$$

on $\mathrm{V}(p,k)$ són els k veïns de p (veure 1.13).

Aleshores, sigui $\mathrm{dv}(p)$ la distància veïna (1.12), es calcula r_p com:

$$r_p(k) := \frac{1}{k}\sum_{j=1}^{k} \mathrm{dv}(\text{last V}(p,k))$$

On k és el nombre de veïns propers que es considerin oportuns, $1 \leq k \leq n$. Per $k = 1$, aleshores $r_p = \mathrm{dv}(p)$. Per $k = N$, aleshores $r_p = m$, on m és la mitjana de $\mathrm{dv}(p)$ de tots els punts.

En pseudocodi:

Algorithm 7 Càlcul de radis d'entorn (*points*, k)

1: $n \leftarrow$ **size of** *points*
2: *near_radius* := **list of** n $\mathbb{R}$ numbers
3: **for all** i **in range** $1, \ldots, n$ **do**
4: $\quad$ *near_radius*[i] $\leftarrow \frac{1}{k}\sum_{j=1}^{k} \mathrm{dv}(\text{last V}(points[i], k))$
5: **return** $(points, near\_radius)$

3.3.4 Vectors normals

Donada una superfície regular $S \subset \mathbb{R}^3$ i $N\colon S \to \mathcal{S}^2$ una orientació (veure secció 1.7). Donat un conjunt de punts de la superfície $P = \{p_i \in S\colon i = 1, ..., n \mid n \in \mathbb{N}\}$, es vol trobar uns vectors $n_i \mid i = 1, .., N$ unitaris, que estimin les rectes normal determinades per $N(p_i)$, és a dir,

$$|\langle n_i, N(p_i)\rangle| \approx 1$$

Gràcies al pas anterior, *càlcul de radis d'entorn*, cada punt p té associat un radi r, que determina un entorn de punts de contacte en p. Per tant, el núvol de punts es pot denotar per els parells $(p_i, r_i)_{i=1,\ldots,n}$.

L'estratègia per calcular els vectors n_i serà, per a cada punt p_i, determinar el **pla de mínims quadrats** del conjunt de punts $\mathrm{B_P(p_i, r_i)}$ (veure secció 1.14). El **pla de mínims quadrats** es defineix de la següent manera:

Definició 3.1. Pla de mínims quadrats. Sigui $p_i \in \mathbb{R}^3$ una llista de punts per $i = 1, \ldots, m$, i sigui un pla qualsevol amb vector normal $n \in \mathcal{S}^2$ que passa pel punt $q \in \mathbb{R}^3$, aleshores, el **pla de mínims quadrats** és aquell que minimitza la funció

$$E(n, q) = \sum_{i=1}^{m} ((p_i - q) \bullet n)^2 \tag{3.1}$$

Lema 3.2. Sigui $p_i \in \mathbb{R}^3$ una llista de punts per $i = 1, \ldots, m$, aleshores el **pla de mínims quadrats** de (p_i) passa per el punt mitjà de (p_i). Es pot trobar una demostració en el següent article [10].

Utilitzant el lema 3.2, es pot expressar l'equació 3.1 com:

$$E(n) = \sum_{i=1}^{m} (\hat{p}_i \bullet n)^2 \tag{3.2}$$

on els punts $\hat{p}_i = p_i - center$ s'han traslladat a de forma que el seu punt mitjà sigui l'origen de coordenades. D'aquesta forma el pla de mínims quadrats passarà per l'origen.

L'equació es pot escriure en forma matricial com:

$$E(n) = \|X^T n\|^2 \tag{3.3}$$
$$\text{on } X = [p_1, ..., p_m] \in \mathbb{R}^{3\times m}$$

Per minimitzar $E(n)$ s'utilitzarà el següent teorema:

Theorem 3.3. Descomposició del valor singular (DVS). Sigui $A \in \mathbb{R}^{m\times n}$ una matriu qualsevol, aleshores existeixen matrius $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ ortogonals, i $\hat{S} \in \mathbb{R}^{m\times n}$, tal que

$$A = U \cdot \hat{S} \cdot V^T, \quad \hat{S} := \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix}$$
$$\text{on } S = \mathbf{diag}(\sigma_1, \ldots, \sigma_r)$$

tal que $\sigma_1 \geq \ldots \geq \sigma_r$ són valors únics anomenats els **valors singulars**, $r \leq \min m, n$ és el rang de A. Les primeres r columnes de U s'anomenen els **vectors singulars** de l'esquerra de A, i les primeres r columnes de V s'anomenen els **vectors singulars** de la dreta de A.

Lema 3.4. Sigui $A \in \mathbb{R}^{m\times n}$ una matriu qualsevol, i $A = U \cdot \hat{S} \cdot V^T$ la seva descomposició en valors singulars. Aleshores AA^T admet una descomposició en valors propis de la forma:

$$A \cdot A^T = U \cdot \Lambda \cdot U^T$$
$$\text{amb } \Lambda = \hat{S}\hat{S}^T = \mathbf{diag}(\sigma_1^2, \ldots, \sigma_r^2, 0, \ldots, 0) \text{ és una matriu } m \times m$$

els valors propis de AA^T són iguals als valors propis al quadrat de A, i els vectors propis de AA^T són els vectors singulars de l'esquerra de A.

Proposition 3.5. Sigui $E(n) = \|X^T n\|^2$ el problema del pla de mínims quadrats. Aleshores el vector n que minimitza la funció $E(n)$, és el **vector singular de l'esquerra** de X (de la seva descomposició DVS) amb el mínim valor singular associat [12].

Demostració. Usant el Teorema de DVS, existeix matrius $U \in \mathbb{R}^{3\times 3}$, $V \in \mathbb{R}^{m\times m}$ ortogonals, i una matriu $\hat{S} \in \mathbb{R}^{3\times m}$ tal que $X = U\hat{S}V^T$. La matriu $\hat{S}$ és de la forma:

$$\hat{S} = \left[\begin{array}{ccc|c} \lambda_1 & 0 & 0 & \\ 0 & \lambda_2 & 0 & \mid 0_{3\times(m-3)} \\ 0 & 0 & \lambda_3 & \end{array}\right]$$

on λ_1, λ_2, λ_3 són els valors singulars de X.

D'aquesta forma:

$$X^T n = V\hat{S}^T U^T n = Vq$$

$$\text{on } q = \hat{S}^T U^T n = \begin{bmatrix} \lambda_1\langle u_1, n\rangle \\ \lambda_2\langle u_2, n\rangle \\ \lambda_3\langle u_3, n\rangle \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Donat que V és ortogonal, $\|Vx\| = \|x\|\ \forall x \in \mathbb{R}^m$. Per tant:

$$\|X^t n\|^2 = \|Vq\|^2 = \|q\|^2 = \lambda_1^2\langle u_1, n\rangle^2 + \lambda_2^2\langle u_2, n\rangle^2 + \lambda_3^2\langle u_3, n\rangle^2$$

Donat que els vectors u_1, u_2, u_3 són les columnes de U, i U és una matriu ortogonal de $\mathbb{R}^{3\times 3}$, els vectors u_i formen una base ortogonal de $\mathbb{R}^3$. Per tant existeixen $\mu_1, \mu_2, \mu_3 \in \mathbb{R}$ tal que:

$$n = \mu_1 u_1 + \mu_2 u_2 + \mu_3 u_3$$

Suposant que λ_1 és el valor singular de mòdul més petit, aleshores es té:

$$\begin{aligned} \|X^t n\|^2 &= (\lambda_1\mu_1)^2 + (\lambda_2\mu_2)^2 + (\lambda_3\mu_3)^2 \\ &= \lambda_1^2(\mu_1^2 + \mu_2^2 + \mu_3^2) + (\lambda_2 - \lambda_1)\mu_2 + (\lambda_3 - \lambda_1)\mu_3 \\ &= \lambda_1^2 + (\lambda_2 - \lambda_1)\mu_2 + (\lambda_3 - \lambda_1)\mu_3 \\ &\geq \lambda_1^2 \end{aligned}$$

Per tant, $X^T n$ queda minimitzat si s'escull $n = u_1$ on u_1 és el vector singular de l'esquerra de X amb el valor singular λ_1 de mòdul menor.

□

Utilitzant el lema 3.4 i la proposició 3.5, es pot trobar el vector n del pla de mínims quadrats, buscant els valors i vectors propis de XX^t que, al ser una matriu de $\mathbb{R}^{3\times 3}$, existeixen mètodes senzills de càlcul de valors i vectors propis. Per tant, l'algorisme del càlcul de normals per un punt d'un núvol de punts és el següent:

Algorithm 8 Càlcul de radis d'entorn (*points*, *near_radius*)

1: $n \leftarrow$ **size of** *points*
2: *normals* := **list of** n $\mathbb{R}$ numbers
3: **for all** i **in range** $1, \ldots, n$ **do**
4: $\quad$ *near_points* $\leftarrow \mathrm{B}_P(points[i], near\_radius[i])$
5: $\quad$ $X \leftarrow$ **matrix of** *near_points* **as columns**
6: $\quad$ $A \leftarrow X \cdot X^T$
7: $\quad$ $e\_values, e\_vectors \leftarrow \text{EigenDecomposition}(A)$
8: $\quad$ $normals[i] \leftarrow$ **select** v **from** *e_vectors* **with minimal** *e_value*
9: **return** $(points, near\_radius, normals)$

Per a la la funció *EigenDecomposition* de l'algorisme, s'ha usat la formula presentada en [21] per a matrius 3×3. Però també es podria haver utilitzat la formula de resolució d'equacions cúbiques estàndard.

3.4 Visualització

Un cop acabada la fase de *Processament de dades*, comença la fase de *Visualització*. El núvol de punts, després de la fase de *Processament de dades* és de la forma $(p_i, c_i, r_i, n_i)_{i=1,\ldots,n}$, on p_i denota la posició, c_i denota el color, r_i denota el radi d'entorn i n_i denota el vector normal.

Com ja s'ha explicat en la secció 2.5, per l'algorisme de visualització s'aplicarà el mètode *Splatting* 2.4.1, però lleugerament modificat. La principal modificació és escalar el kernel de convolució en funció del valor r_i per així adaptar la convolució a la irregularitat de la mostra.

A partir de la funció gaussiana com a kernel d'interpolació, s'utilitza la següent funció com a footprint:

Definició 3.6. Funció footprint per la visualització

$$gauss\_footprint(x, y) = \exp -\frac{\|(x, y)\|^2}{0.2}$$

És a dir, s'utilitza la mateixa funció per tots els punts. Per escalar la mida del $foot\_print$ en funció de r_i es defineix el següent procés:

Definició 3.7. Projecció aproximada d'una esfera Sigui (T_P, T_Δ) una càmera, i sigui p un punt del núvol qualsevol amb radi r. L'objectiu es calcular una aproximació de la projecció de càmera d'una esfera de centre p i radi r.

Suposant $g = T_\Delta(p)$, es calculen els punts

$$right = T_P(g + (r, 0, 0))$$
$$left = T_P(g - (r, 0, 0))$$

aleshores, una aproximació de l'esfera projectada es pot determinar amb l'esfera de centre $\hat{p}$ i radi $\hat{r}$:

$$\hat{p} = (T_P \circ T_\Delta)(p)$$
$$\hat{r} = right_x - left_x$$

Amb aquest càlcul determina correctament l'esfera projectada quan la càmera és de tipus ortogràfic, però no en el cas que la càmera sigui de tipus perspectiu. En cas que la càmera tingui perspectiva, es considerarà com a una "aproximació".

Amb aquest càlcul, per un punt en coordenades de pantalla (x, y) es pot calcular l'efecte que rep del footprint per el punt p de forma:

$$gauss\_footprint(\frac{1}{R}((x, y) - (\hat{p}_x, \hat{p}_y)))$$

Així doncs, l'algorisme de la fase de visualització basat en *Splatting*, és el següent:

Algorithm 9 Splatting Adaptat ($Image$, $Camera(T_P, T_\Delta)$, $Lights$, $PointCloud(position, color, radius, normal)$, $gauss\_footprint$)

sort elements **in** *PointCloud* **by** distance to camera (from back to front)
for all $position, color, radius, normal$ **in** *PointCloud* **do**
  $color \leftarrow$ **Phong**$(point, normal, Camera, Lights)$
  $\hat{p} \leftarrow T_\Delta(position)$
  $right \leftarrow \hat{p} + (radius, 0, 0)$
  $left \leftarrow \hat{p} - (radius, 0, 0)$
  $pixel_{right} = (k \circ \pi \circ T_P)(right)_i$
  $pixel_{left} = (k \circ \pi \circ T_P)(left)_i$
  $R \leftarrow pixel_{right} - pixel_{left}$
  $(i, j) \leftarrow (k \circ \pi \circ T_{Camera})(position)$
  for $\hat{i}$ **in range** $i - R, \ldots, i + R$ **do**
    for $\hat{j}$ **in range** $j - R, \ldots, j + R$ **do**
      $coords \leftarrow 1(\frac{\hat{i}-i}{R}, \frac{\hat{j}-j}{R})$
      $a \leftarrow gauss\_footprint(coords)$
      $Image[\hat{i}, \hat{j}] \leftarrow color \cdot a + (1 - a) \cdot Image[\hat{i}, \hat{j}]$
return *Image*

Capítol 4

Desenvolupament dels software

4.1 Introducció

En aquest capítol s'explicaran les parts més importants del desenvolupament del programari aplicant la solució teòrica proposada al capítol 3. Aquest capítol està dividit en 3 seccions: *Anàlisi* 4.2, *Disseny* 4.3 i *Optimització* 4.4

- En la secció d'anàlisi (4.2), es determinem els requisits que ha de tenir el programari. S'explicarà quina plataforma de treball s'ha escollit pel desenvolupament i perquè. I finalment s'introduirà, a grans trets, l'arquitectura de la llibreria gràfica escollida en la plataforma de treball.
- En la secció de disseny (4.3) es mostra, de forma simplificada, els components que formen el nostre programa, quines funcions tenen, i quins pertanyen a les fases *Processament de dades* (3.3) i *Visualització* (3.4), formulades al capítol 3.
- Finalment, en la secció d'optimització (4.4), es detallen les qüestions més importats que s'han tingut en compte per tal de d'implementar les fases *Processament de dades* i *Visualització* amb el mínim cost de computació possible.

4.2 Anàlisi

4.2.1 Requisits del programa

- **Característiques de les dades**
  - S'han de llegir núvols de punts formats per un vector posició i, opcionalment, i color **RGB**.
- **Paràmetres configurables**

  - Oferir dos tipus de càlcul de **visibilitat**:
    * **Directe**, és a dir, sense convolució.
    * ***Splatting***, aplicant convolució,
  - Oferir dos tipus d'**il·luminació**:
    * **Directe**, sense fons d'il·luminació ni ombrejat.
    * ***Phong***.
  - Oferir tres tipus de **color font** de a pa visualització:
    * El color per **defecte** en tots els punts.
    * El color **mostrejat** del núvol de punts.
    * El color representant el **vector normal** del núvol de punts.

- **Resposta del sistema**
  - El refresc de visualització ha de ser igual o superior a 30 FPS. És a dir, el temps de generació d'una imatge ha de ser inferior a 30^{-1}s.
  - Oferir a l'usuari control per poder canviar en temps real la càmera de visualització i els paràmetres de **visibilitat**, **il·luminació** i **color font**.

4.2.2 Plataforma de treball

Per desenvolupar el programari s'ha decidit utilitzar el llenguatge *C++* per el codi base del programa i la api *OpenGL* per la configuració de la *GPU*, que utilitza el llenguatge *GLSL* per a la programació de *shaders*. Per a gestionar les finestres i inputs de l'usuari, s'ha utilitzat la llibreria *GLFW*. I per a estructures i funcions matemàtiques s'ha utilitzat la llibreria *GLM*.

El projecte s'ha desenvolupat mitjançant *Visual Studio 2019*. S'ha procurat que el codi sigui *cross-platform* entre Windows, Linux i Mac.

4.2.3 Arquitectura de OpenGL

La *GPU* és la unitat encarregada d'executar algorismes de visualització per generar imatges. La majoria de *GPU*s estan dissenyades i optimitzades per executar algorisme de visualització de tipus projectiu (2.2.3). Aquest és un dels motius per el qual s'ha escollit el mètode *Splatting*.

OpenGL és una llibreria per configurar i programar algorismes de tipus projectiu a la *GPU*. El tipus de model 3D que processa OpenGL són col·leccions de primitives geomètriques, que poden ser: punts, línies, o triangles. Per tant, una malla de triangles o un núvol de punts serien models compatibles.

OpenGL divideix un procés de visualització en 5 grans fases:

- *Vertex Shader*: Aquesta fase és programable per l'usuari i és la fase on es produeixen els càlculs a nivell de vèrtex del model. Aquesta fase, generalment, ha de projectar els vèrtexs a coordenades normalitzades de càmera, i determinar les dades de cada vèrtex que s'utilitzaran en les següents fases (com per exemple el color del punt, radi d'entorn, etc).

- *Generació de primitives*: Aquesta fase és automàtica, i consisteix en agrupar les primitives amb els vèrtexs que la generen. Per un núvol de punts, aquesta fase s'ignora.

- *Rasterització*: Com s'ha explicat a la secció 2.2.3, aquesta fase s'ocupa de determinar els píxels on projecten cada primitiva.

- *Fragment Shader*: Aquesta és la segona fase programable per l'usuari. Aquí es fan els càlculs necessaris per a cada píxel de forma individual (també es coneix com a *Píxel Shader*).

- *Test i barreja*: L'ultima fase, és el test de profunditat per a cada píxel (per tal de descartar píxels amagats), i la barreja de colors per a punts diferents que van a parar en el mateix píxel. Aquesta fase no és programable, però *OpenGL* ofereix eines per a configurar-la amb les necessitats de l'usuari.

En la figura 4.1 es mostren les fases d'execució de *OpenGL*.

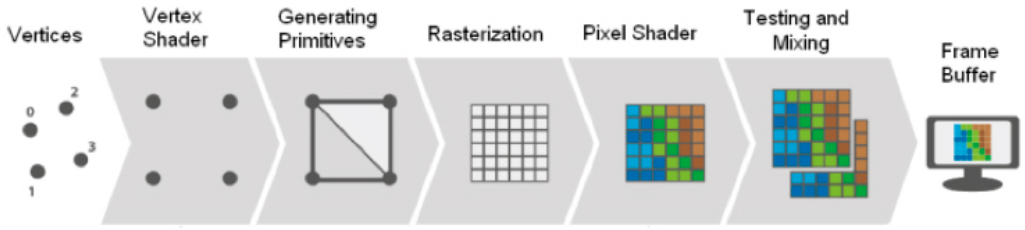


Figura 4.1: Pipeline de OpenGL

Un programa de renderització en *OpenGL*, s'anomena *Shader Program* i està compost, generalment, per dos petits programes: un pel *Vertex Shader* i un pel *Fragment Shader*.

La característica que permet executar programes de visualització eficientment, és en la concurrència de cada fase. És a dir, la fase *Vertex Shader* s'executa en paral·lel per al màxim nombre de vèrtexs possibles, i la fase de *Fragment Shader* s'executa en paral·lel per al màxim nombre de píxels. Aquest fet també dificulta l'execució d'algorismes que depenguin de l'ordre, que per exemple el model de transparències, en aquests casos s'hauran d'aplicar mètodes alternatius o optimitzacions.

4.2.4 Splatting en OpenGL

Com s'ha vist en la secció anterior 4.2.3, *OpenGL* ofereix una bona estructura per executar un algorisme basat en *Splatting*. Per tal d'executar un mètode de tipus *Splatting*, es poden programar les fases de *Vertex Shader* i *Fragment Shader* de la següent manera:

- *Model*: Es configura *OpenGL* perquè tracti el model com a una col·lecció de punts individuals, on cada punt és una tupla d'una posició, color, radi, i vector normal.
- *Vertex Shader*: En aquesta fase es produeix el càlcul de projecció de càmera a coordenades normalitzades per a cada posició del vèrtex. I cal també determinar la mida del rectangle de píxels, que rasteritzarà *OpenGL* per a cada vèrtex. En aquesta fase també es pot aplicar el model d'il·luminació *Phong* a partir del vector normal i fonts d'il·luminació definides per l'usuari.
- *Fragment Shader*: En aquesta fase és necessari tenir un càlcul del $footprint$, el qual, en cas que sigui constant, es pot definir com a una taula de valors precalculada. Llavors, a partir de la coordenada del píxel relativa al rectangle rasteritzat, es determina l'efecte del $footprint$ i es calcula el color.
- *Barreja*: L'última fase s'ha de configurar per sumar els colors a cada píxel aplicant el model de transparències definit a 2.12. Com que la formula de transparències requereix que els punts estiguin ordenats en la profunditat, caldrà d'aplicar mètodes d'optimització per permetre la màxima concurrència.

4.3 Disseny

En la figura 4.2 es mostren els components que formen l'aplicació i amb les relacions d'oferta-requeriment d'interfícies.

4.3.1 Mòdul *Processament de dades*

El mòdul de *Processament de dades* està compost per dos components:

- *Arbre 3D*: conté una estructura i funcions de cerca, per calcular de forma eficient els veïns pròxims d'un punt i l'entorn de punts (com s'han descrit en la secció 1.9). En la secció 4.4.1 s'explica en més detall el mètode de cerca en l'arbre.
- *Núvol de punts*: és l'estructura que conté les dades del núvol de punts, i tots els sub-mètodes del *Processament de dades*. Aquest component ofereix una interfície a l'*Aplicació* i al *Motor Gràfic* per a que puguin accedir a les dades del núvol.

4.3.2 Mòdul *Visualització*

El mòdul de *Visualització* està compost per tres components:

- *Càmera*: modela la posició, direcció, zoom i projecció d'una càmera, i ofereix mètodes per calcular les seves transformacions (descrites a la secció 1.28).
- *Shader*: engloba totes les crides de la llibreria *OpenGL*. Aquest component representa una classe per diversos tipus de programes de shaders, que són escollits a través de la interfície que ofereix al *Motor Gràfic*.
- *Motor Gràfic*: conté el bucle principal de la visualització i les variables dels paràmetres de visualització (descrits en 4.2.1) que són actualitzades a través de l'input de l'usuari i s'envien a la classe del *Shader*.

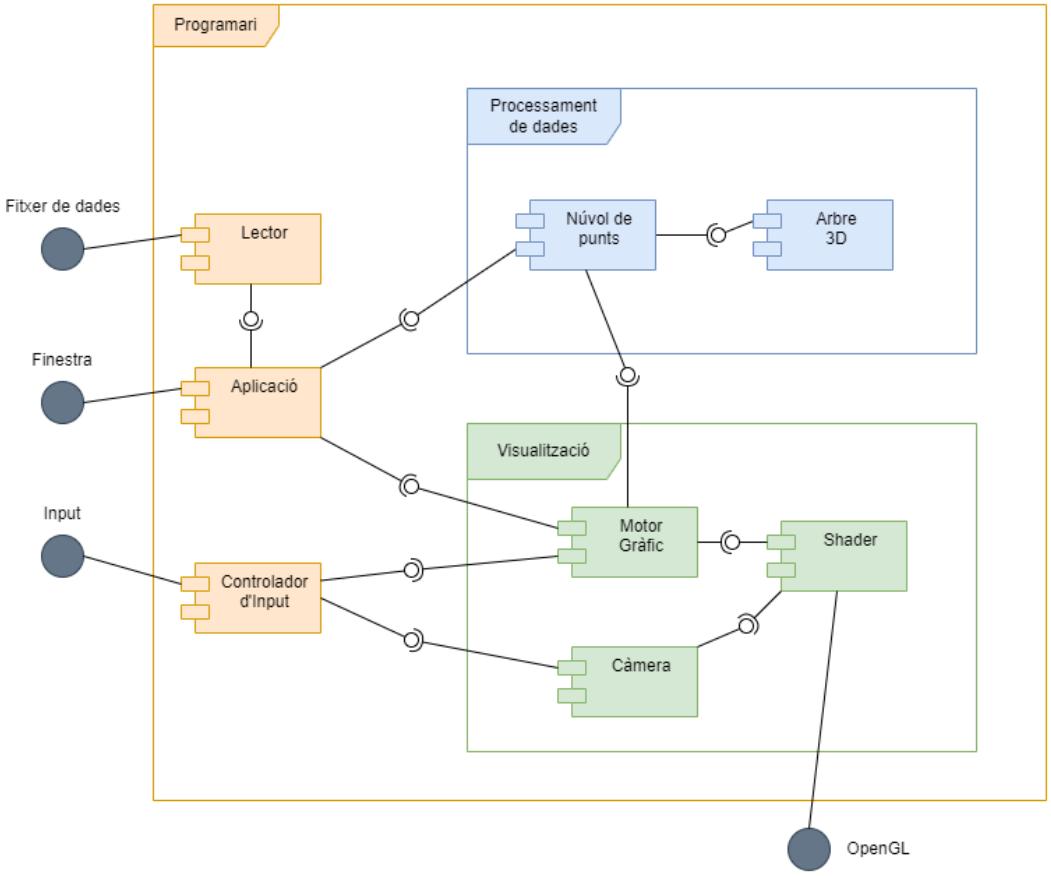


Figura 4.2: Diagrama UML de components del programa. La connexió cercle denota que el component ofereix una interfície. La connexió arc denota que el component requereix una interfície

4.3.3 Aplicació

L'*Aplicació* és el component principal del programa. En iniciar-se una sessió, aquest component processa els paràmetres inicials, carrega les dades del núvol de disc i les envia al

component *Núvol de punts* perquè siguin processades. Un cop finalitzada el processament, envia el núvol de punts al *Motor Gràfic* i engega la visualització.

4.4 Optimització

4.4.1 Arbre 3D

Com es mostra en el Capítol 3, en el *Processament de dades* s'utilitzen contínuament les funcions de veïns pròxims $V_P(p,k)$, i entorns de punts $B_P(p,r)$ (definides en la secció 1.9). Si s'utilitza un algoritme de força bruta per calcular aquestes funciones, aleshores tenen una complexitat lineal en la mida del núvol $O(n)$ per a un sol punt. Com que es necessita repetir el càlcul per a cada punt del núvol, aleshores es té una complexitat de l'ordre de $O(n^2)$.

Per tal de reduir la complexitat, s'utilitzarà una estructura per organitzar els punts de tal manera que ens permeti el càlcul de $V_P(p,k)$ i $B_P(p,r)$ amb una complexitat menor de l'ordre de $O(n \log n)$. Aquesta estructura s'anomena *Arbre 3D* (o en k dimensions: *Kd Tree* [22])

Un *Arbre 3D* organitza una col·lecció de punts en una estructura d'arbre binari, on cada node representa un punt de la col·lecció i genera, implicitament, una partició binària de l'espai a partir d'un pla perpendicular a un eix de coordenades i que passi per el punt del node.

Millora

Per a la solució del treball s'ha escollit una millora del *Arbre 3D* convencional, anomenada *Revised kd Tree*, extreta de [9]. La millora es basa en acotar cada node en una caixa ortogonal als eixos, de forma que tots els punts dels sub-nodes estiguin dins de la caixa. D'aquesta forma, en el procés de cerca es podrà descartar branques de l'arbre més ràpidament degut a una millor acotació de la distància mínima.

A més d'això, també s'ha afegit una millora pròpia per el cas de buscar veïns propers de punts que pertanyin a l'arbre. El que es fa és crear els nodes de l'arbre de manera que es puguin accedir mitjançant una llista indexada, i assegurar-se que l'index del node i l'index del punt que representa el node són el mateix. Això permet que, en lloc de començar totes les cerques des de l'arrel de l'arbre, es pugui començar la cerca des del propi node del punt. I llavors s'explora l'arbre com si fos un *graph*, és a dir, explorant pares i fills. Per tal de tenir una cota de distància quan s'exploren nodes pares, el que es fa és crear una altre caixa ortogonal en cada node. Però aquest cop la caixa no engloba els punts del nodes fills, sinó que representa la màxima caixa possible sense englobar la resta de punts que no siguin fills.

Estructura d'un node

Un node està format per:

- *pointIndex* := index del punt que representa el node
- *point* := el punt que representa el node
- *parent* := node pare
- *left* := node fill esquerra
- *right* := node fill dret
- *minBounds* := mínima caixa que engloba els punts fills
- *maxBounds* := màxima caixa que no engloba la resta de punts que no son fills

Construcció

La construcció de l'arbre es descriu en l'algorisme 10.

Algorithm 10 Construcció de l'arbre

```
1:  procedure CreateSubtree(list of points pointList, list of nodes nodeList, list of integers
    pointIndexes, integer depth, Box outsideBox)
2:      axis ← depth mod 3
3:      medianIndex ← select i from pointIndexes so that pointList[i] is the median of
    sublist pointList[pointIndexes]
4:      node ← nodeList[medianIndex]
5:      node.pointIndex ← medianIndex
6:      node.insideBox ← Box(pointList[node.pointIndex])
7:      node.outsideBox ← outsideBox
8:      leftBounds := bounds ∩ left partition of axis
9:      rightBounds := bounds ∩ right partition of axis
10:     leftIndexes := points in pointList before median
11:     rightIndexes := points in pointList after median
12:     node.left ← left := CreateSubtree(pointList, nodeList, leftIndexes, depth+1, left-
    Bounds)
13:     node.right ← right := CreateSubtree(pointList, nodeList, rightIndexes, depth+1,
    rightBounds)
14:     node.insideBox ← node.insideBox ∪ left.insideBox ∪ right.insideBox
15:     return
```

Cerca

La funció per cercar els K veïns d'un punt del núvol es descriu en l'algorisme 11. La funció per cercar $B_P(p,k)$ és molt semblant a l'algorisme 11, només canviat la condició de parada, així que no es considera necessari descriure l'algorisme.

Algorithm 11 K veïns propers

```
1:  procedure NearestK(tree nodeList, integer index, integer K)
2:      create results as list of points
3:      create searchQueue as list of (currentNode, prevNode)
4:      node := select index from nodeList
5:      position := node.position
6:      worstKDist := +∞
7:      insert (node, none) in searchQueue
8:      while not searchQueue empty do
9:          currentNode, prevNode := select and remove front from searchQueue
10:         currentDist := distance(position, currentNode.location)
11:         if size of results < K then
12:             insert currentNode.location in results results
13:         else if currentDist < worstKDist then
14:             remove point in results with worst distance from position
15:             insert currentNode.location in results
16:         worstKDist ← worst distance(position, p) for all p in results
17:         left := currentNode.left
18:         right := currentNode.right
19:         parent := currentNode.parent
20:         if left ≠ prev and distance(left.insideBox, position) < worstKDist then
21:             insert (left, currentNode) in searchQueue
22:         if right ≠ prev and distance(right.insideBox, position) < worstKDist then
23:             insert (right, currentNode) in searchQueue
24:         if parent ≠ prev and distance(current.outsideBox, position) < worstKDist then
25:             insert (parent, currentNode) in searchQueue
        return results
```

4.4.2 Visualització per plans

Per tal d'oferir una visualització amb baixa tassa de refresc, s'ha de programar l'algorisme de visualització de forma que maximitzi els recursos de la *GPU*. Com s'ha explicat en la Secció 4.2.3, la *GPU* està dissenyada per executar els shaders en paral·lel, però el mètode teòric de visualització descrit en el Capítol 3 requereix una execució en sèrie i de forma ordenada.

En aquesta secció es presenta una adaptació de la solució teòric,a que ofereix una fidelitat d'imatge lleugerament inferior, però compensa per altre banda degut a la gran

millora en rendiment. El mètode es divideix en 2 passos: *Divisió per plans*, *Layered Splat*. La idea és agrupar el núvol de punts en talls generats per plans paral·lels a l'eix z de la càmera. Aleshores, per ordre de pla més proper al més llunyà de càmera, s'aplica una versió de *Splatting* per a cada tall individual. La imatge resultant de cada pla es va composant amb l'anterior seqüencialment.

Aquest procés permet dues millores d'eficiència: una és que la versió de *Splatting* per a un únic tall es pot executar concurrentment, ja que, al haver una profunditat semblant per a cada punt, es poden composar les petjades de forma commutativa; la segona millora és que en el moment de composar imatges, es poden marcar els píxels que ja tinguin opacitat màxima, i evitar que repeteixi càlculs innecessaris per aquests píxels en talls següents.

A continuació es descriu en més detall cada pas.

Divisió per plans

En aquest pas es vol dividir el núvol de punts en m talls generats per $m+1$ plans equidistants paral·lels a l'eix z de càmera. Però, es desitja una visualització en temps real i interactiva, on la càmera es pot moure dinàmicament. Això fa que el cost de ordenar els punts dels núvols a través d'un eix sigui massa alt. La solució que es proposa es predefinir una sèrie de eixos [1] i pre-calcular els m talls de mida uniforma. Aleshores, en cada iteració de visualització, s'escollirà l'eix de les divisions en funció de quin s'acosta més al vector director de càmera. S'ha de tenir en compte que els talls han d'estar ordenats en funció de més proper a més llunyà a càmera. Per tant, es possible que en algun moment es tinguin que invertir la llista dels talls (quan la direcció a càmera sigui oposada a la direcció d'un eix).

Layered Splat

Sigui (c_i, α_i) per $i = 1, \ldots, n$, una mostra de colors i opacitats d'un mateix pla de profunditat, definim el el color $c^{'}$ i alpha $\alpha^{'}$ resultants de la mescla com:

$$c^{'} = \frac{\sum_{i=1}^{n} c_i \cdot \alpha_i}{\sum_{i=1}^{n} \alpha_i}$$

$$\alpha^{'} = \max(\sum_{i=1}^{n} \alpha_i \ , \ 1)$$

És a dir, es sumen els colors c_i com a una ponderació normalitzada utilitzant els α_i com a pes. La $\alpha^{'}$ resultant és la suma dels α_i, tenint en compte que el màxim valor d'opacitat és 1. Aquesta formula és la modificació principal que s'aplica al mètodede *Splatting* per tal de permetre concurrència.

El mètode optimitzat de visualització *Splatting* es mostra en l'algorisme 12, que utilitza els algorismes 13 i 14.

[1] per exemple es poden definir 12 eixos corresponents a les cares d'un Rhombicuboctahedron [14]

Algorithm 12 Splatting per nivells

```
1: procedure LayeredSplat(list of slices sliceList, Camera camera, width, height)
2:     assert sliceList is sorted front to back by  camera z aixs
3:     Image frame := Image of size width×height
4:     Matrix stencil := Matrix of size width×height
5:     for all slice in sliceList do
6:         sliceImage ← SplatLayer(slice, camera, width, height, stencil)
7:         frame ← Compose(frame, sliceImage, stencil)
8:     return frame
```

Algorithm 13 Splatting per un tall

```
1: procedure SplatLayer(list of vertex vertexList, Camera camera, width, height, stencil)
2:     Image frame := Image with opacity of size width×height
3:     Image sumColor := Image of size width×height
4:     Matrix weight := Matrix of size width×height
5:     for all vertex in vertexList do (concurrently)
6:         vertex.color ← Phong model
7:         vertex.position ← projection(camera, vertex.position)
8:         for all (pixel, α) in footprint(vertex.position, vertex.size) do (concurrently)
9:             sumImage[pixel] ← sumImage[pixel] + α· vertex.color
10:            weight[pixel] ← weight[pixel] + α
11:    for all pixel in frame do (concurrently)
12:        frame[pixel][color] = sumImage[pixel] / weight[pixel]
13:        fram[pixel][alpha] = max(weight[pixel], 1)
14:    return frame
```

Algorithm 14 Composició d'imatges

```
1: procedure Compose(Image with opacity frame, Image with opacity slice, stencil)
2:     for all pixel in frame do (concurrently)
3:         (f_color, f_alpha) ← frame[pixel]
4:         (s_color, s_alpha) ← slice[pixel]
5:         frame[pixel][color] ← f_color · f_alpha + s_color · (1 − f_alpha)
6:         frame[pixel][alpha] ← 1 − ((1 − f_alpha) · (1 − s_alpha))
7:     return frame
```

Capítol 5

Simulacions i resultats

5.0.1 Test de rendiment de l'Arbre 3D

Per el test de rendiment s'ha comparat el mètode base de *Revised Kd Tree* [9] denotat per *NORMAL REVISED*, contra la modificació del mètode introduïda en aquest treball 4.4.1, que es denota per *INDEX BASED REVISED*. S'ha comparat el temps que tarden en calcular la distància veïna per a tots els punts d'una mostra. Els resultats han sigut els següents:

Punts	**NORMAL REVISED**	**INDEX BASED REVISED**
10k	114 ms	51 ms
100k	2214 ms	1039 ms
500k	8484 ms	3812 ms
1000k	18120 ms	7991 ms

Taula 5.1: Comparació de rendiment

El següent test de rendiment és comparar el temps de divisió per plans per un eix, utilitzant l'Arbre 3D ja construït, contra el temps que tarda la funció *sort* de c++ a ordenar el núvol de punts per un eix. Els resultats han sigut els següents:

Punts	**TREE BASED SLICING**	**PLAIN SORTING**
10k	115 ms	13 ms
50k	373 ms	96 ms
100k	614 ms	228 ms
500k	1910 ms	996 ms
1000k	3377 ms	1890 ms

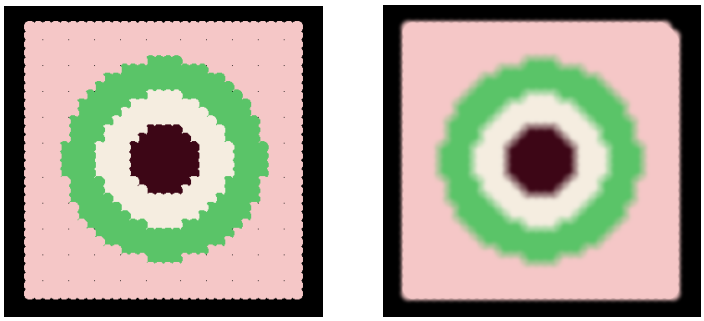
Taula 5.2: Comparació de rendiment

Conclusions

La modificació proposada per el càlcul de la distància veïna és considerablement efectiu segons es veu a la Taula 5.1. Per altre banda, el Kd-Tree no resulta eficient quan es desitja una partició de la mostra per en una gran quantitat de plans, en aquests casos ordenar la mostra directament és més òptim 5.2.

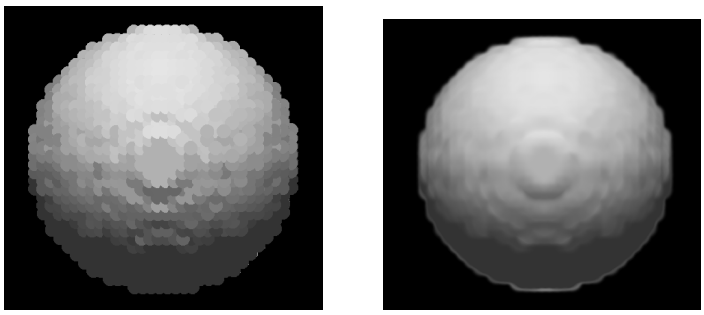
5.0.2 Reconstrucció d'una superfície

A continuació es mostra una comparació visual de dues funcions de taques diferents: una taca binària i una taca gaussiana. Totes les imatge has sigut generades per l'algorisme desenvolupat.



(a) (b)

Figura 5.1: Comparativa 1: (a) taca binària, (b) taca gausiana



(a) (b)

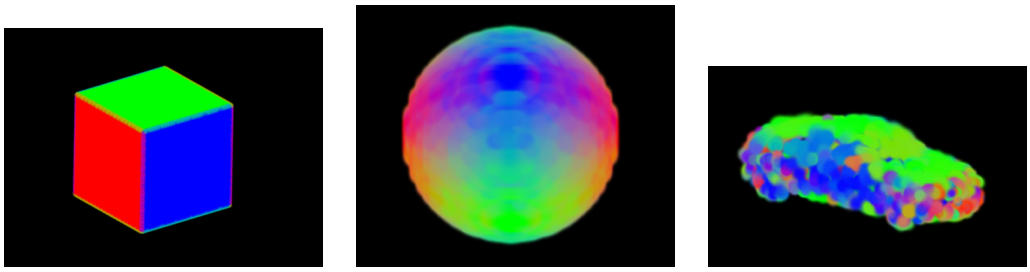
Figura 5.2: Comparativa 2: (a) taca binària, (b) taca gausiana

(a) (b)

Figura 5.3: Comparativa 3: (a) taca binària, (b) taca gausiana

5.0.3 Aproximació de vectors normals

En les següents imatges es visualitza el vector normal com un color *RGB*. Es pot observar com en núvols de punts uniformes, dona un millor càlcul del vector normal que per núvols irregulars.



(a) (b) (c)

Figura 5.4: (a) i (b) són núvols regulars, (c) un núvol irregular

5.0.4 Validacions del resultats

Primer de tot, remarcar que tots els objectius de desenvolupament definits en el capítol s'han assolit satisfactòriament. Els requisits del programa que s'han determinat en el Capítol 4, secció 4.2 també s'han aconseguit. El que resta és valorar, de forma qualitativa, el mètode específic de visualització, en concret, cal valorar la qualitat de reconstrucció del núvol de punts i el càlcul de normals. Es pot dir que els resultats arriben al llindar de ser vàlids, però hi ha força espai de millora. La convolució que s'ha aplicat en el mètode desenvolupament té una pèrdua significant de "nitidesa" sobretot en els punts de gran cobertura. També cal destacar que l'algorisme no és (actualment) apte per a núvols de punts molt grans. S'ha determinat empíricament que el màxim nombre de punts visualitzables interactivament ronda l'ordre dels trenta mil. El més és positiu és que, com s'explicarà en el següent Capítol, l'algoritme és molt extensible, en el sentit que existeixen molts camins clars de millora.

Capítol 6

Conclusions and Future work

6.1 Conclusions

L'algorisme *Splatting* (descrit en el Capítol 2), tot i ser un mètode inventat l'any 1991, en ser un mètode de tipus projectiu s'adapta força bé a les arquitectures de les *GPU* modernes. A més, gràcies al fet que ofereix una reconstrucció contínua a partir de la mostra discreta del núvol de punts, és el mètode per a núvols de punts amb la millor relació qualitat-cost. El factor més limitant pel que fa a l'eficiència del mètode, és la necessitat de compondre les *petjades* de cada punt amb el model de transparències, ja que aquest depèn de l'ordre de composició, i, per tant, no permet un càlcul concurrent de les petjades en cada píxel. Tot i això, s'ha vist que hi ha maneres d'evitar lleugerament aquesta limitació i, ben segur, que encara hi ha més millores per descobrir.

6.2 Feina futura

Petjada

La petjada es projecta a pantalla a través d'una aproximació ortogonal (com s'ha explicat en la secció 3.4), per tant, en cas que la càmera tingui una projecció de perspectiva, aleshores l'efecte de la petjada no serà el correcte. Com a millora de l'algorisme, s'hauria de considerar o bé un mètode diferent en el cas que la càmera estigui en perspectiva, o bé un mètode general per a tota mena de càmeres.

Concurrència

Per tal d'oferir un nivell de concurrència superior, es poden explorar dos camins:

- Oferir una operació de composició de petjades que sigui independent en l'ordre. Això es coneix com el problema de *Order Independent Transparency* [18] (OIT en curt).

Una possible extensió seria aplicar un mètode de OIT ja existent, o bé crear-ne un de nou específic per a núvols de punts.

- Fer una visualització en dues passades, on la primera serveixi per eliminar els punts que es considerin amagats per punts més propers. Aleshores, la segona visualització ja no caldrà aplicar el model ordenat de transparències, ja que tots els punts restants es poden considerar que pertanyen al mateix pla de profunditat.

Procés del núvol dinàmic

Una millora seria en simplificar la quantitat de punts del núvol, ajuntat múltiples punts prou propers, en un únic punt mitjà que contindrà la mitjana de tots els valors col·lapsats. Aquesta operació de simplificació es pot aplicar, per exemple, en funció de la distància a càmera.

Apèndix A

Manual tècnic

A.1 Instal·lació del programa

El programa ofereix un fitxer *CMAKELists.txt* per tal de compilar-lo còmodament tant a Windows com a Linux. L'única dependència és tenir la llibraria OpenGL instal·lada. Si es té Windows es pot utilitzar *Visual Studio 2019* ja que incorpora compatibilitat amb cmake

A.2 Ús del programa

Hi ha dos comandes per executar el programa:

- ./pointdraw.exe <primitiva>
  On primitiva pot ser:
  - slice
  - slice-noisy
  - star
  - star-noisy
  - sphere-surface
  - cube-solid-opaque
- ./pointdraw.exe -f <fitxer>
  On fitxer pot ser:
  - pointclouds/airplane_1.ply
  - pointclouds/car_1.ply

Els principals controls de la càmera i paràmetres són:

- Boto dret: Girar càmera
- WASD: traslladar càmera
- Roda ratolí: Zoom
- CONTROL + Roda ratolí: Increment/Decrement mida de punts
- FLETXA AMUNT - FLETXA AVALL: canvi de font de colors
- X: canvi de kernel de convolució (binari o gaussià)
- TAB: activació/desactivació il·luminació amb Phong.

Bibliografia

[1] https://favpng.com/png_view/colorful-cubes-rgb-color-model-visible-spectrum-color-space-png-8ZFeMqEc.

[2] https://yearbook.com/support/article/image-resolution-explained/.

[3] https://www.ideamoocs.com/courses/orthographic-projections/.

[4] https://learnopengl.com/Getting-started/Coordinate-Systems.

[5] http://scarletsky.github.io/2020/06/10/games101-notes-rasterization/.

[6] https://en.wikipedia.org/wiki/Phong_reflection_model#/media/File:Phong_components_version_4.png.

[7] Scratchapixel 2.0, *Rasterization: a practical implementation*, https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/overview-rasterization-algorithm, 01/2022.

[8] Song Ho Ahn, *Opengl projection matrix*, http://www.songho.ca/opengl/gl_projectionmatrix.html, 01/2022.

[9] Yewang Chen, Lida Zhou, Yi Tang, Jai Puneet Singh, Nizar Bouguila, Cheng Wang, Huazhen Wang, and Jixiang Du, *Fast neighbor search by using revised k-d tree*, Information Sciences **472** (2019), 145–162.

[10] Emil Ernerfeldt, *Fitting a plane to many points in 3d*, http://www.ilikebigbits.com/2015_03_04_plane_from_points.html, 01/2022.

[11] Stephanie Glen, *Z-score: Definition, formula and calculation*, https://www.statisticshowto.com/probability-and-statistics/z-score/, 01/2022, From StatisticsHowTo.com.

[12] Jerry Greenough, *Least squares plane fit*, http://www.jgxsoft.com/examples/Plane%20Fitting/Plane%20Fitting.html, 01/2022.

[13] Ignasi Mundet i Riera, *Apunts: Geometria diferencial de corbes i superfícies*, Universitat de Barcelona, 2020.

[14] Cool Math 4 Kids, *The rhombicuboctahedron*, https://www.coolmath4kids.com/more/polyhedra/polyhedra-rhombicuboctahedron, 01/2022.

[15] Mariano Trebino Lopez, *The phong illumination model*, https://mtrebi.github.io/2017/01/25/phong-illumination.html, 01/2022.

[16] Bruce MacEvoy, *Do primary colors exist?*, http://www.handprint.com/HP/WCL/color6.html#imaginary, 01/2022.

[17] _____, *Modern color models*, http://www.handprint.com/HP/WCL/color7.html, 01/2022.

[18] Morgan McGuire and Louis Bavoil, *Weighted blended order-independent transparency*, Journal of Computer Graphics Techniques **2** (2013), no. 4.

[19] National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center, *Lidar 101: An introduction to lidar technology, data, and applications*, Revised. Charleston, SC: NOAA Coastal Services Center, 2012.

[20] John Pawasauskas, *Volume visualization with ray casting*, https://web.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p1/ray-cast.htm, 01/2022.

[21] Oliver K. Smith, *Eigenvalues of a symmetric 3 × 3 matrix*, Commun. ACM **4** (1961), no. 4, 168.

[22] Carnegie Mellon University, *Kd trees*, https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/kdtrees.pdf, 01/2022.

[23] Lee Alan Westover, *Splatting: a parallel, feed-forward volume rendering algorithm*, Ph.D. thesis, The University of North Carolina at Chapel Hill, 1991.

[24] Wolfram Math World, *Convolution*, https://mathworld.wolfram.com/Convolution.html, 01/2022.