TREBALL FINAL DE GRAU

Realtime Softbody Physics using Position Based Dynamics

Miquel Valverde Parera
Grau en Disseny i Producció de Videojocs

CURS 2020 - 21

TecnoCampus
Escola Superior Politècnica

Centre adscrit a la

upf. Universitat Pompeu Fabra Barcelona

TecnoCampus

Centres universitaris adscrits a la

upf. Universitat Pompeu Fabra Barcelona

Grau en Disseny i Producció de Videojocs

Realtime Softbody Physics using Position Based Dynamics

Miquel Valverde Parera

Tutor: Marco Antonio Rodríguez Fernández

TecnoCampus Mataró-Maresme

Abstract

This project is focused on studying the main real time mesh deformation simulation techniques when a physical impact happens in a 3D videogame. It is well known that PBD is the state of the art of real time physical simulations, corroborated by *Nvidia* among other sources. The state of the art is deeply analysed and PBD is putted in practice inside *Unity* engine.

Resum

L'objectiu d'aquest treball és l'estudi de les principals tècniques per a simular deformacions de malles a temps real quan es produeix un impacte físic en un videojoc 3D. És ben sabut que la tècnica PBD conforma l'estat de l'art de les simulacions físiques a temps real, corroborat per *Nvidia* entre altres fonts. S'analitza a fons l'estat de l'art i es posa en pràctica la tècnica PBD mitjançant el motor *Unity*.

Resumen

El objetivo de este trabajo es el estudio de las principales técnicas para simular deformaciones de mallas a tiempo real cuando se produce un impacto físico en un videojuego 3D. Es bien sabido que la técnica PBD conforma el estado del arte de las simulaciones físicas a tiempo real, corroborado por *Nvidia* entre otras fuentes. Se analiza a fondo el estado del arte i se pone en práctica la técnica PBD mediante el motor *Unity*.

INDEX GENERAL

INDEX DE FIGURES

INDEX DE TAULES

GLOSSARI DE TERMES

FEM	Mètode numèric per resoldre equacions diferencials de forma aproximada.	*Finite Element Method*
FPS	Número d'imatges mostrades consecutivament per pantalla en la durada d'un segon.	*Frames Per Second*
GPU	Processador extern dedicat al processament d'imatges.	*Graphics Processing Unit*
KNN	Algoritme estadístic de classificació basat en la distància entre punts.	*K-Nearest Neighbours*
LBS	Idea de transformar vèrtex d'una malla basat en múltiples transformacions lineals.	*Linear Blend Skinning*
PBD	Tècnica de simulació de sistemes de partícules.	*Position Based Dynamics*

1 Introducció

Aquest treball estudia les diferents tècniques per a realitzar deformacions físiques a temps real en videojocs.

La motivació principal és l'impacte que el realisme ha demostrat tenir en els videojocs (Zibrek et al., 2019), en conjunt amb el mercat *in crescendo* de la indústria dels videojocs a nivell global a l'actualitat (Michaud, 2016). Això impulsa l'evolució de hardware que permet sistemes cada cop més complexos i realistes.

La motivació conseqüent resideix en l'absència total de motors de jocs que integrin funcionalitats per a realitzar deformacions no rígides de sèrie. Aquest fet es deu a que algunes propietats físiques no es tenen en consideració a l'hora de dissenyar les mecàniques de joc habituals (Maggiorini et al., 2014). Aquesta limitació afecta als *game designers* alhora d'imaginar i projectar noves mecàniques.

L'objectiu consisteix en identificar les tècniques canòniques emprades en videojocs de fama mundial, i implementar una tècnica avalada en nombrosos articles acadèmics escrits per Matthias Müller, actual *Physics research lead* de *Nvidia*, per a simular cossos tous.

Els passos a seguir són l'estudi de l'estat de l'art de les diferents tècniques, i posteriorment la implementació pràctica en el motor *Unity* de la tècnica *Position Based Dynamics*, a partir d'ara PBD.

2 Objectius

2.1 Objectius Principals

- Identificar les tècniques existents per a simular deformacions físiques a temps real en videojocs.

- Analitzar en profunditat la tècnica PBD per a la simulació de cossos tous de manera eficient, i estable.

2.2 Objectius Secundaris

- Implementar una simulació física de cossos tous mitjançant la tècnica PBD en el motor *Unity*, tal que aprofiti els recursos del propi motor en la mesura del possible, sense llibreries dinàmiques de tercers.

3 Marc Teòric

Actualment, hi ha dos camps principals de recerca per a representar simulacions físicament acurades. Els dos tenen en comú l'ús d'elements finits, i que són aproximacions discretes a la realitat.

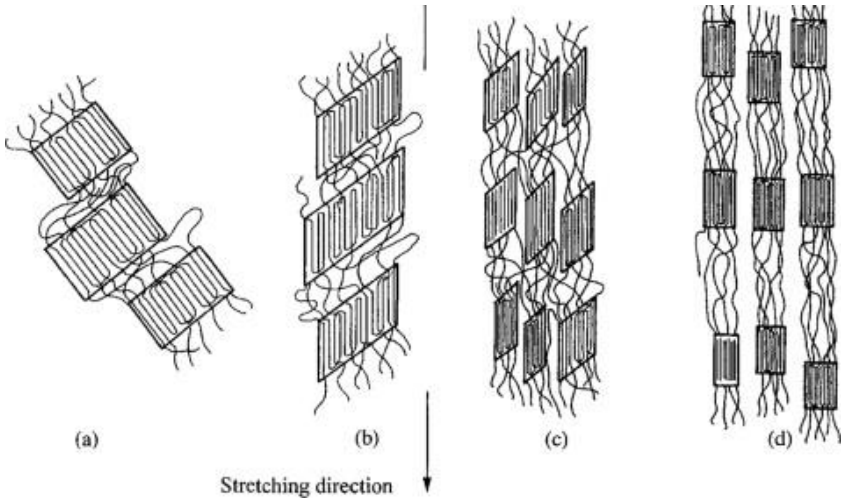
La simulació de sistemes físics és un repte computacional en gràfics interactius, com ho són els videojocs, ja que no únicament han de reproduir el model físic, sinó que també han de ser robusts i eficients.

3.1 Deformació plàstica

Una deformació plàstica és la deformació permanent o canvi de forma d'un cos rígid sense fractura sota l'acció d'una força sostinguda.

Tal com diu tal com diu Aifantis (1987), una deformació plàstica, com qualsevol altre procés físic, es pot entendre millor considerant i analitzant adequadament els mecanismes subjacents responsables. Per tal de simular-lo a gran escala només ens centrem en els mecanismes més importants i simples de tots, els moviments de dislocació.

Figura 1. Deformació plàstica d'un polímer semi-cristal·lí



Font: Mercier et al., 2002

A la Figura 1 podem veure làmines connectades per cadenes amorfes abans i després de produir-se una deformació plàstica a nivell microscòpic i esquemàtic. Aquesta és la idea fonamental on resideixen les solucions i tècniques basades en físiques per a simular el comportament d'aquests cossos de manera aproximativa.

3.2 Finite Element Method (FEM)

Finite Element Method és una tècnica que prové de la mecànica estructural, de la qual Hrenninkoff (1940) se'n considera el fundador. FEM és una tècnica computacional originalment utilitzada per a resoldre problemes de flexió de plaques (Clough, 1960). Aquesta tècnica ha trobat lloc en molts camps de la simulació computacional en física i enginyeria. Actualment, una de les aplicacions més conegudes és per simular proves de xoc virtuals a NCAPs (van Ratingen et al., 2016).

A través d'aquesta tècnica, es poden simular molts altres escenaris, com les interaccions entre òrgans del cos humà. En aquest article tenim un exemple que analitza com afecten els moviments pèlvics en un context de radioteràpia de càncer de pròstata (Boubaker et al., 2009).

Aquestes simulacions no són a temps real, la precisió de les simulacions és molt elevada i no és factible l'aplicació en videojocs, a no ser que es simplifiquin considerablement. Això explica l'absència majoritària, però no absoluta, de referents interactius lúdics, analitzats més endavant.

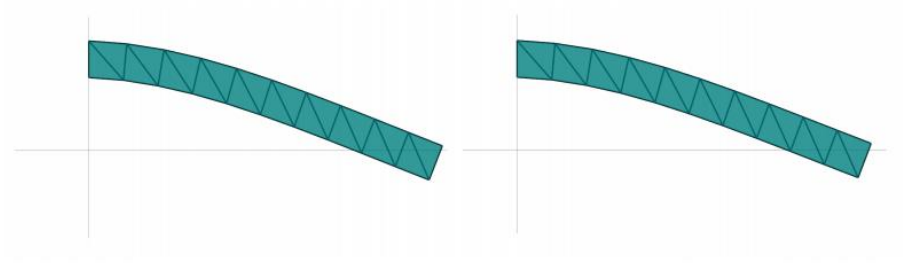
3.3 Position Based Dynamics (PBD)

3.3.1 Introducció

Position Based Dynamics (Tsai, 2017) és una tècnica ràpida, estable i controlable que s'utilitza sovint en la indústria dels videojocs i coneixem com a *SoftBody*.

Matthias Müller, *lead* de recerca de físiques de *Nvidia*, assegura que els mètodes tradicionals *Finite Element Methods* (FEM) poden ser reformulats utilitzant *Extended Position Based Dynamics* (XPBD) (Macklin et al., 2016) amb una precisió visualment indistingible entre els dos mètodes.

Figura 2. Voladís modelat amb elements FEM.
Esquerra: Newton Solver. Dreta: XPBD.



Font: Macklin et al., 2016

3.3.2 Integració

Un sistema de partícules E és un conjunt de partícules que està definit per un conjunt de normes que descriuen un comportament independent per a cada partícula pertanyent a ell $P_n \in E$.

Per descriure les forces físiques externes que apliquen en un sistema de partícules és necessari utilitzar mètodes numèrics d'integració per equacions diferencials ordinàries.

Existeixen diferents mètodes per integrar l'equació de moviment conservatiu de Newton, però aquest treball només es centra en la integració de Verlet, 1967.

La velocitat de Verlet implica calcular de forma implícita la velocitat v de les partícules, tal que:

$$x_{i+1} = x_i + v_i \Delta t$$

$$v_{i+1} = v_i + a_i \Delta t$$

On Δt és el diferencial de temps, sovint conegut com *Delta Time* en videojocs.

On a és l'acceleració de la segona llei de Newton.

On x és la posició de les partícules.

Podem representar la velocitat aproximada de forma implícita:

$$x_{i+1} = 2x_i - x_{i-1} + a_i \Delta t^2$$

Aquest mètode és estable per l'aplicació en videojocs, però en altres contextos s'utilitzen mètodes d'integració més robusts, com el de Euler amb les seves variants implícites o explícites.

3.3.3 Sub-Stepping

Com exposen els estudis de Macklin et al., 2019, s'ha pogut observar que prendre tants passos (*steps*) com el pressupost de temps Δt_f ho permeti, és molt més eficient que realitzar més iteracions per aconseguir la convergència del sistema.

Per comptes d'integrar amb el diferencial de temps Δt_f, s'ha vist que és una pràctica molt més eficaç integrar amb $\Delta t_f / n$.

$$\Delta t_s = \frac{\Delta t_f}{n_{steps}}$$

On n és el nombre de *steps*.

On Δt_s és el diferencial resultant de utilitzar *sub-steps*.

On Δt_f són els *sub-steps* permesos per al fotograma actual, i pot variar en funció dels FPS.

A continuació es mostra un bucle on es mostra l'ordre d'execució de cada tasca de la simulació realitzant *sub-stepping*:

Figura 3. Pseudocodi que aplica Sub-Stepping

Algorithm 1 Substep XPBD simulation loop

perform collision detection using $\mathbf{x}^n$, $\mathbf{v}^n$
$\Delta t_s \Leftarrow \frac{\Delta t_f}{n_{steps}}$
while $n < n_{steps}$ **do**
  predict position $\tilde{\mathbf{x}} \Leftarrow \mathbf{x}^n + \Delta t_s \mathbf{v}^n + \Delta t_s^2 \mathbf{M}^{-1} \mathbf{f}_{ext}(\mathbf{x}^n)$
  for all constraints **do**
    compute $\Delta\lambda$ using Eq (7)
    compute $\Delta\mathbf{x}$ using Eq (4)
    update $\lambda^{n+1} \Leftarrow \Delta\lambda$ (optional)
    update $\mathbf{x}^{n+1} \Leftarrow \Delta\mathbf{x} + \tilde{\mathbf{x}}$
  end for
  update velocities $\mathbf{v}^{n+1} \Leftarrow \frac{1}{\Delta t_s}\left(\mathbf{x}^{n+1} - \mathbf{x}^n\right)$
  $n \Leftarrow n + 1$
end while

Font: Macklin et al., 2019

Tal com descriu l'article, el *sub*-stepping no suposa una sobrecàrrega de restriccions (*constraints* en anglès), els quals són estudiats amb més profunditat en el proper apartat.

3.3.4 Restriccions

Les restriccions (*constraints* en anglès) són aquell conjunt de normes que defineixen un sistema de partícules. A continuació s'analitza en més profunditat basat en les teories existents, les quals van aparèixer entorn al mateix temps (Stam, 2009; Müller, 2008).

Un *constraint* $j \in [1, \ldots, M]$ consisteix en:

- Una funció $C_j: R^{3n_j} \rightarrow R$ capaç de modelar qualsevol superfície a través del teorema de *simplicial complexes* de Brouwer (1910).
- Amb una cardinalitat n_j que equival al nombre de partícules al que aplica al mateix temps aquella restricció.
- Un set d'índex $\{i_1, \ldots, i_{n_j}\}$, $i_k \in [1, \ldots, N]$ de les partícules en funció de la cardinalitat n_j.
- Un paràmetre per definir la rigidesa $k_j \in [0 \ldots 1]$.

La satisfacció d'un *constraint* j pot ser bilateral o unilateral. Aquest tipus d'igualtat o desigualtat descriu la tendència del *constraint*, on la rigidesa és definida per k_j.

3.3.4.1 Projecció

La projecció de *constraints* és essencialment necessària per satisfer el sistema. Es pren el conjunt de restriccions $j \in [1, \ldots, M]$ com a un sistema d'equacions no lineals el qual pot resoldre's de diferents formes (Milaszewicz, 1987).

La tècnica PBD resol el sistema d'equacions no lineal mitjançant linealització local, el qual consisteix en una iteració no lineal de Gauss-Seidel.

Gauss-Seidel demostra ser més robust i té una millor convergència del sistema. Malauradament però, no es pot computar de forma paral·lela de forma senzilla, essent un desavantatge sobre aquesta opció respecte Jacobi.

També és comú projectar sobre el sistema mitjançant mètodes de Jacobi. Aquest mètode és factible d'aplicar de forma paral·lela processant cada *constraint* o partícula. S'ha pogut observar com la convergència utilitzant aquest mètode és més lenta, i menys robusta (Bridson et al., 2002).

3.3.4.2 Restricció de Distància

Per tal de simular un comportament de relaxació gradual, s'aplica un *constraint* que descriu:

$$C(p_1, p_2) = |p_1 - p_2| - d$$

El qual pot ser desenvolupat de forma general:

$$\Delta_{p_1} = -\frac{w_1}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|}$$

$$\Delta_{p_2} = +\frac{w_2}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|}$$

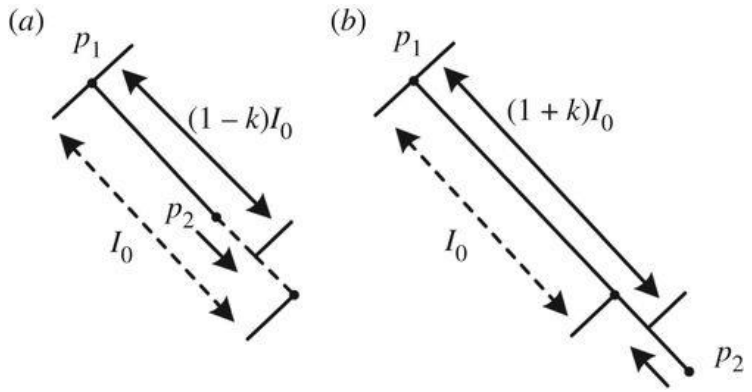
On Δ_{p_1} i Δ_{p_2} són conseqüència de la cardinalitat $n_j = 2$.

On w_1 i w_2 són les masses inverses de cada partícula.

On p_1 i p_2 són la posició de les partícules.

On d és la distància entre partícules.

Figura 4. Sobre estirament Mass-Spring



Font: Xu et al., 2018

Aquest principi que parteix de la llei de Hook, permet simular sistemes de teixits tous de qualsevol magnitud, formats per sistemes de partícules.

3.3.4.3 Restricció de Col·lisió

Quan s'assumeix la detecció d'una col·lisió, podem donar una resposta establint un *constraint* unilateral tal que:

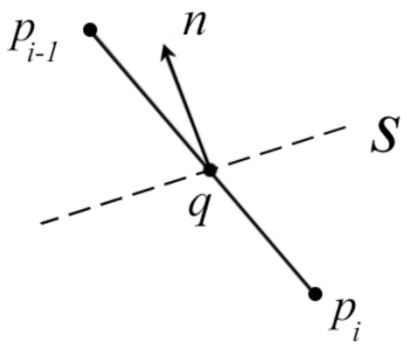
$$C(p) = (p_{i-1} - q)^T n$$

On p és la posició de la partícula,

On q és el punt de penetració de la partícula, a la superfície S.

On n és la normal de la superfície S.

Figura 5. Restricció de col·lisió



Font: Elaboració pròpia

La restricció està satisfeta quan $C(p) \geq 0$, és a dir quan la partícula p_i estigui per sobre de la normal de la superfície. Mentre aquesta condició es compleix, no s'aplica aquesta restricció.

3.3.5 Skinning

Un aspecte representatiu d'aquesta tècnica per facilitar la simulació a temps real, és la capacitat d'adaptar-se amb tècniques d'animació esquelètica*, i.e. Skinning* (Aburumman & Fratarcangeli, 2015) i funcions d'interacció a través de geometria simplificada com FFD (McDonnell & Qin, 2007) o *Cage-Based* com es mostra en aquest article d'un ex-professor del Tecnocampus (García et al., 2013).

D'aquesta manera podem disminuir la complexitat lineal d'aplicar un comportament físic per cada partícula, i només afectar un casc simplificat que envolta la malla real amb un núvol de punts.

En aquest TFG, s'utilitza *Linear Blend Skinning* (LBS) per a adaptar-se amb el motor *Unity*. Aquesta tècnica descriu una equació a la que estan supeditats tots els vèrtex de la malla tal que:

$$v'_i = \sum_{j=1}^{m} w_{i,j} T_j v_i = \left(\sum_{j=1}^{m} w_{i,j} T_j \right) v_i$$

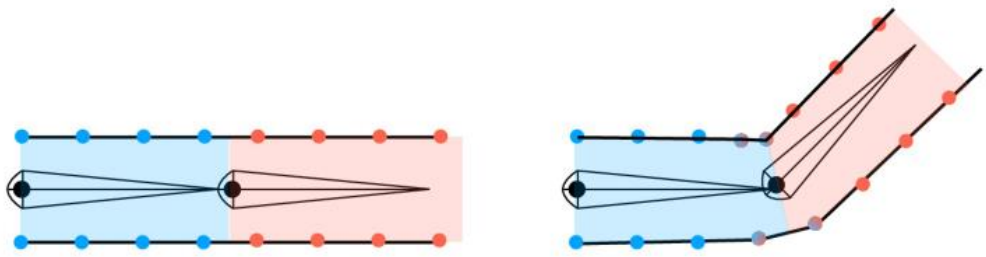
On v_i és l'ièssim vèrtex.

On $w \in [0 \ldots 1]$ és un escalar que actua com a pes de l'os j que actua sobre el vèrtex i.

On T és una transformació matricial.

LBS transforma els vèrtex d'una malla amb una combinació lineal de transformacions de l'os respectiu.

Figura 6. Esquerra: articulació original, Dreta: articulació afectada



Font: Transparències de Stanford CS248, 2019

A la Figura 6, les zones on s'ajunten els ossos, s'estableix un pes $w = 0.5$.

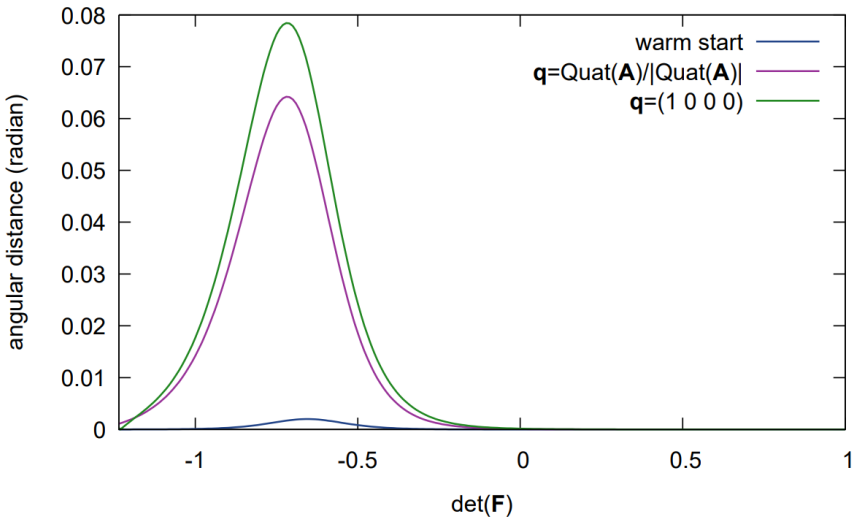
3.3.5.1 Whaba Problem

Un dels problemes a l'hora de simular deformacions en un sistema de partícules, és la necessitat de extreure la part rotacional d'una matriu arbitrària 3×3. Aquesta necessitat pot deure's a diverses raons fonamentals, com ara aplicar tècniques de *Shape Matching* (Matthias Müller et al., 2016) o LBS entre d'altres.

Aquest és un problema conegut de matemàtiques aplicades, primerament proposat per Grace Whaba al 1965. El qual pretén trobar la matriu de rotació entre dos sistemes de coordenades que parteixen d'un conjunt vectorial amb pes. Una de les tècniques més populars per resoldre aquest problema, és utilitzar descomposició polar (Farrell et al., 1966).

En un article escrit per Matthias Müller et al., (2016) s'exposa un mètode robust per resoldre aquest problema de forma iterativa, que demostra recórrer la mínima distància angular:

Figura 7. Distància angular, des de diferents inicis



Font: Matthias Müller et al., 2016

Aquest mètode és ideal per a GPUs degut a que no presenta branques, és curt i adaptable en qualsevol context, tal com conclou l'article.

4 Referents

Trobem obres de gran influència que utilitzen algunes de les tècniques objecte d'estudi en aquest TFG. A continuació es mostren alguns referents històrics o avantguardistes que utilitzen cada una de tècniques segons les seves necessitats i capacitats.

4.1 Star Wars: The Force Unleashed

Tal com es mostra a la conferència de la *Games Developer Conference* (*GDC*, 2009), *Star Wars: The Force Unleashed* (2008), utilitza l'element finit per a la simulació a temps real de deformacions, essent un dels pocs referents interactius que opta per una tècnica físicament acurada. En el transcurs de la conferència s'exposen les dificultats per optimitzar la tècnica.

És important observar com aquest videojoc, publicat al 2008, no utilitza PBD (Müller, 2008), ja que l'estudi va ser publicat al mateix any sense possibilitat de donar marge a l'estudi i implementació d'aquesta nova tècnica, la qual hagués resultat més òptima pel cas d'estudi.

4.1 Hitman: Codename 47

Els algorismes desenvolupats per *Hitman: Codename47* (2000) per *IO Interactive* formen part d'un sistema de físiques responsable pel moviment de roba, plantes i altres tipus de cossos tous. El més destacat d'aquest referent és la simulació de la caiguda de cossos morts, que depèn d'on reben l'impacte de la bala, i interactuen amb l'entorn físicament (Jakobsen, 2001). A l'article es detalla el plantejament per enfrontar-se a tots aquests reptes, essent un dels referents exemplars per aquest treball, i per molts videojocs que han sortit posteriorment.

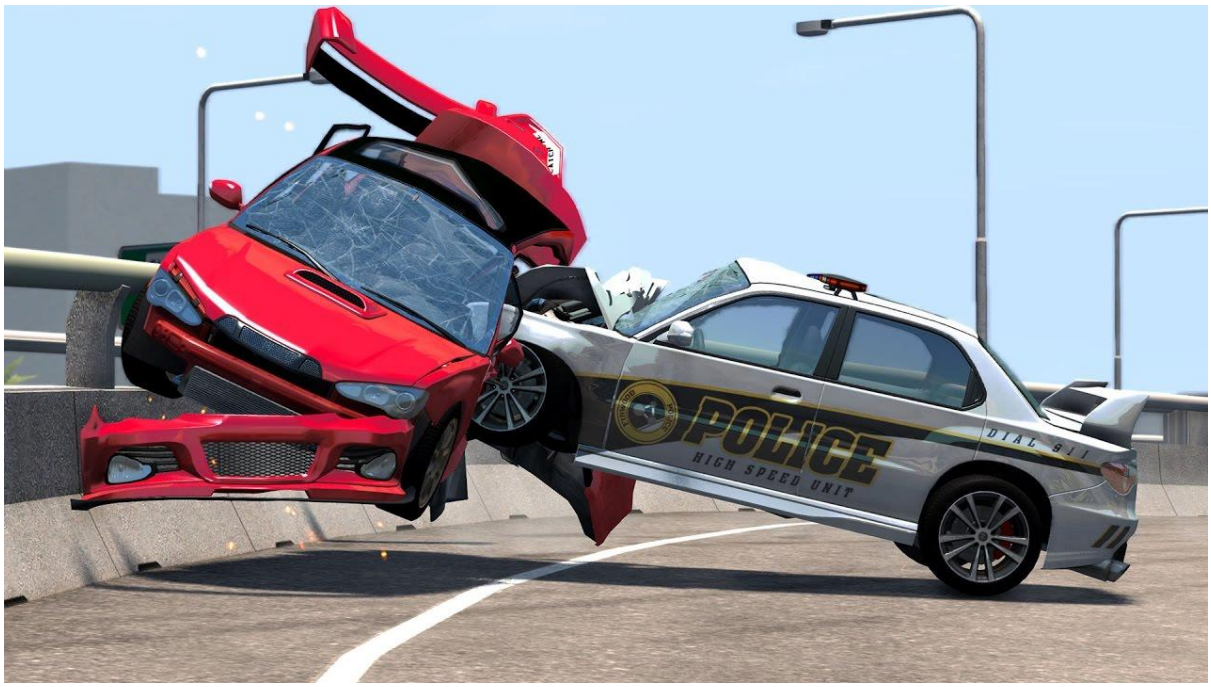
4.2 BeamNG

BeamNG.drive (2015) és un estudi independent, referent exemplar i pioner que utilitza *Position Based Dynamics* per a la deformació físicament acurada de vehicles tal com

es demostra a *JBeam Physics Theory*, 2015. El fet que es tracti d'un estudi independent explica el cost elevat de R+D+I en aquest tipus de tecnologia.

Un altre simulador de físiques que utilitza aquesta tècnica és *Rigs of Rods,* aquest és d'un interès particular en aquest TFG perquè ha esdevingut *open sour*ce darrerament. És considerat l'antecedent de *BeamNG.drive*, ja que *BeamNG* ha contractat desenvolupadors per a incorporar-los en la seva empresa degut a l'experiència adquirida en un projecte anàleg (Reilly, 2012).

Figura 8. Físiques interactives de cossos tous inter-vehiculars.



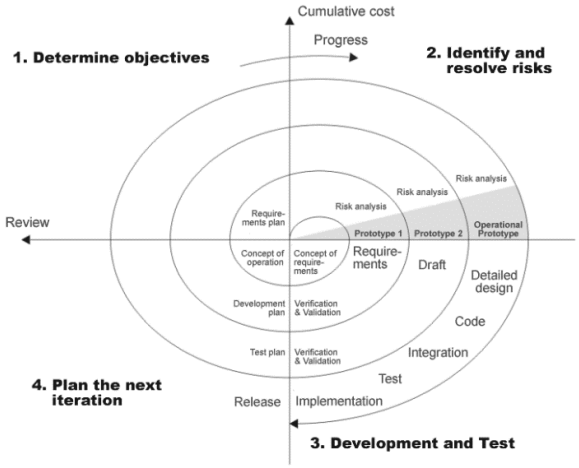
Font: Fotograma de BeamNG, 2015

5 Disseny metodològic i cronograma

5.1 Metodologia

El pla per autogestionar l'organització i garantir la realització del projecte consisteix en un sistema iteratiu de producció, aplicant el Model en Espiral (Boehm, 1986). Aquest model consisteix en iterar donant voltes en espiral, on cada volta equival a una funcionalitat acabada, i en aquest TFG equival a cada fita del producte.

Figura 9. Cicle de vida en espiral



Font: Model proposat per Bohem

Es contemplen diferents etapes que composen la metodologia del projecte:

- La primera etapa es basa en la recerca de l'estat de l'art, la qual està inclosa dins l'avantprojecte, en conjunt amb la planificació del desenvolupament del producte.

- La següent consisteix en el desenvolupament del producte en base a l'etapa prèvia. Aquesta segona etapa està dividida entre les cinc fites descrites a l'apartat de Funcionalitats. Per cada fita es pretén incorporar una nova funcionalitat, això determinarà que es pugui avançar a la següent fita, o s'hagi d'iterar sobre la mateixa sense permetre avançar. Per tant, el mapa que ens tracem per a l'elaboració d'aquest projecte implica un pas crucial per cada fita assolida.

- La tercera i última etapa és la preparació de la defensa davant del tribunal docent, presentació final de la memòria, i dipòsit de la mateixa.

5.2 Mètode

S'utilitza un client basat en *Git*, que permet integrar *Git-Flow* per a gestionar les branques amb un flux de treball continu amb el controlador de versions. D'aquesta manera es delimita cada iteració, s'agilitza el procés, i es treballa de forma no destructiva.

Per al desenvolupament del producte s'utilitza el motor *Unity* per a diverses raons exposades a continuació:

1. La primera raó és el potencial d'expansió que pot tenir un videojoc, si es subministra amb totes les característiques que *Unity* ofereix. Mentre que si s'utilitza un motor propi, o un motor alternatiu, no hi ha aquest potencial degut al reduït nombre de funcionalitats.

2. El motor *Unity* suposa una posada en pràctica ràpida i directe, amb resultats visibles des del primer moment.

Contradiccions a tenir en compte imposades per l'entorn escollit de desenvolupament:

1. *Unity* utilitza l'estàndard aritmètic de coma flotant IEEE$_{745}$ de 32-bit. Malgrat, és de gran necessitat la doble precisió per a una simulació acurada.

2. És de gran necessitat una bona gestió de memòria si es desitja fer el sistema de simulació de físiques eficient, doncs és més adequat utilitzar un *kernel* de la GPU paral·lelitzable per a cada partícula. *Unity* no permet aquests factors.

S'assumeixen aquestes contradiccions perquè es consideren de menor rellevància pel cas d'ús.

5.3 Funcionalitats

La col·lecció de requeriments i compromisos del producte final es mostra a continuació. D'aquesta manera es pot provar que s'ha complert els requeriments com a eina de comprovació.

Descripció dels requeriments funcionals en format de llista:

- Implementar i portar a la pràctica la tècnica PBD en el motor *Unity.*
    - Utilitzar *Verlet Integration* **per a** estimar els diferencials de posició.
    - Implementar els *Constraints* de posició **per a** obtenir un sistema *Mass-Spring* projectat amb *Gauss-Seidel*.
    - Programar un sistema de detecció i resposta de col·lisions **per a** sobrepassar les restriccions de *PhysX*.
    - Programar un sistema de *Skinning* **per a** aplicar deformacions simplificades a objectes complexos a temps real.
    - Incorporar un lector d'arxius capaç de llegir configuracions de partícules en format JSON **per a** facilitar la càrrega de models.

Pel client final, els requeriments funcionals impliquen l'obtenció d'una simulació de cossos tous dins del motor *Unity*.

Taula 1: Requeriments funcionals

ID	Requeriment	Desitjat	Estimació
1	Integració de Verlet	Si	5
2	Restricció *Mass-Spring*	Si	5
3	Restricció de col·lisió	Si	3
4	*Skinning* de malles i partícules	Si	13
5	Lector de configuracions de partícules	Si	2
6	Interacció per part de l'usuari	No	-
7	Col·lisió entre partícules	No	-

Font: Elaboració pròpia

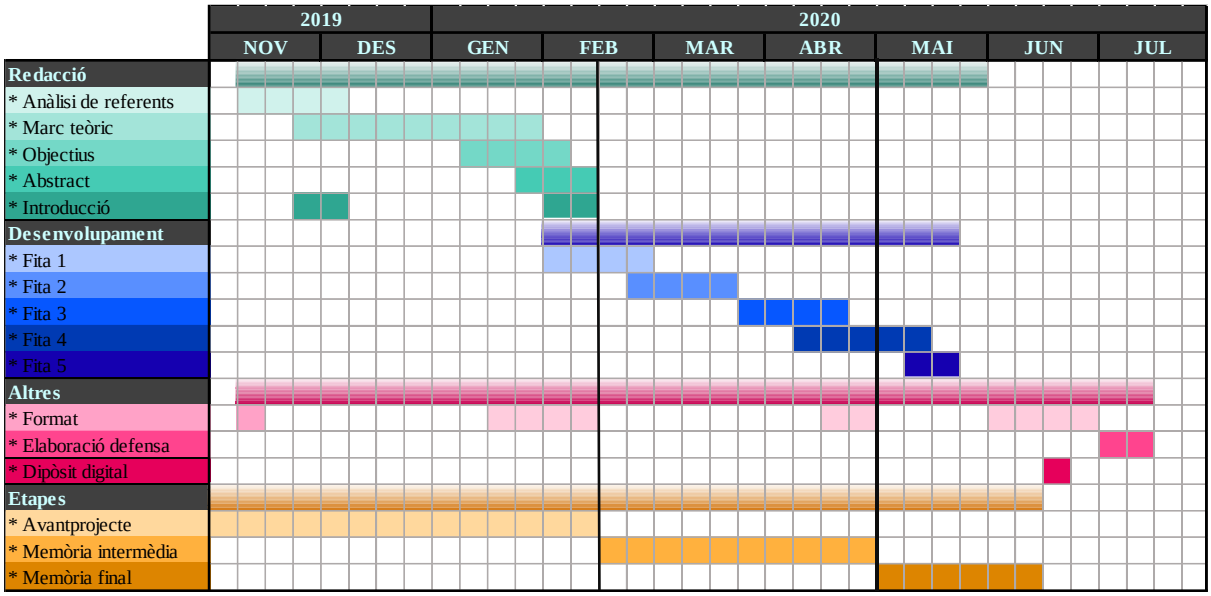
També s'ha assignat l'Estimació *Agile* de Fibonacci per quantificar l'esforç esperat que comporta la complexitat de la tasca. Per últim, s'ha inclòs un conjunt de requeriments no desitjats, que es consideren fora del domini del treball. S'han inclòs

per remarcar quin és el territori que engloba el treball i quin no. S'ha escollit aquests requeriments i no uns altres perquè en un desenvolupament d'escala major, probablement serien els següents passos a seguir, bo i que no els últims.

5.4 Cronograma

En aquest apartat es mostra gràficament la proposta de dates i hores dedicades al projecte. També es pot veure la relació i dependència de tasques. Aquesta planificació s'adapta als lliuraments imposats per la institució.

Taula 2. Metodologia temporalitzada



Font: Elaboració pròpia

La Taula 2 contempla l'abast del projecte amb totes les fases de desenvolupament del producte desglossat.

6 Desenvolupament

El desenvolupament del projecte s'aborda a través d'iteracions que alhora equivalen a les funcionalitats escollides del producte final. L'estructura dels següents subapartats comença amb els objectius de la iteració. Tot seguit s'explica com ha estat la implementació. Per últim, es valora si s'ha aconseguit l'objectiu de la iteració.

6.1 Primera iteració

6.1.1 Objectius de la iteració

La primera iteració consisteix en implementar una arquitectura de classes per facilitar la integració de partícules amb el mètode de *Verlet*.

Es considera conclòs aquest apartat si podem visualitzar el comportament de partícules d'acord amb lleis de la física com són la gravetat. Tot el procés ha d'estar implementat en l'entorn de *Unity* sense recursos de tercers.

6.1.2 Desenvolupament de la iteració

6.1.2.1 Arquitectura

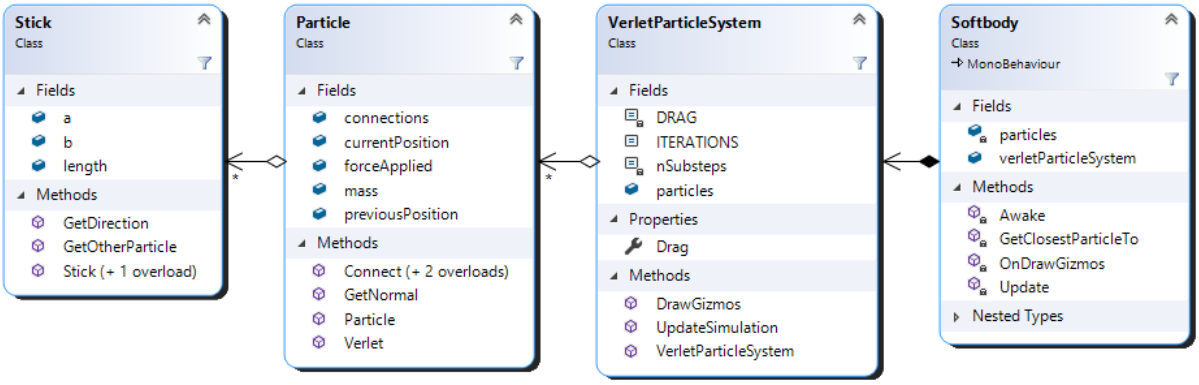
Abans d'implementar el codi, s'elabora un diagrama de classes. Aquest diagrama permet fer un recompte de totes les necessitats per assolir l'objectiu, veure Figura 10. Fer un diagrama de classes permet estructurar objectes, i afronta problemes de forma *Top-Down*.

Top-Down és un estil per afrontar problemes on el producte parteix d'una descripció d'alt nivell sobre què es pretén aconseguir. Es procedeix a trencar les especificacions en trossos més petits fins que el nivell es correspon amb el vocabulari i tipus primitius del llenguatge de programació.

Un cop feta la valoració de tot allò que es vol implementar per assolir l'objectiu, es comença programant aquelles parts que són crucials per tot el desenvolupament

posterior. Concretament el primer que s'aborda és la implementació dels complexes geomètrics.

Figura 10. Diagrama de classes per integrar la simulació



Font: Elaboració pròpia

Per emmagatzemar la informació de l'objecte en una estructura sòlida de classes reutilitzables, es generen quatre nous objectes: *Stick*, *Particle, VerletParticleSystem i Softbody*. Aquests objectes es relacionen a través d'agregació i composició. Aquesta relació permet definir nivells de complexitat més elevats, essent *Softbody* el major exponent i l'únic hereter de *MonoBehaviour* com es mostra al diagrama.

6.1.2.2 Integració

Per integrar la simulació es crea el mètode *VerletParticleSystem*::*UpdateSimulation* que alhora s'encarrega d'integrar cada partícula. Aquesta decisió aplica el patró de responsabilitat de classes, atorgant la feina necessària a l'objecte que correspon.

Com s'ha exposat a la secció de teoria, s'aplica *substepping* que demostra obtenir una simulació més acurada i rígida (Macklin et al., 2019).

En el procés d'integració també es té en compte la massa de la partícula per si es desitja modificar-se, i s'integra en base a la constant d'acceleració de la força de gravetat. *Unity* facilita l'accés des de *Physics.Gravity*, d'aquesta manera podem modificar la constant des de la interfície nativa del motor.

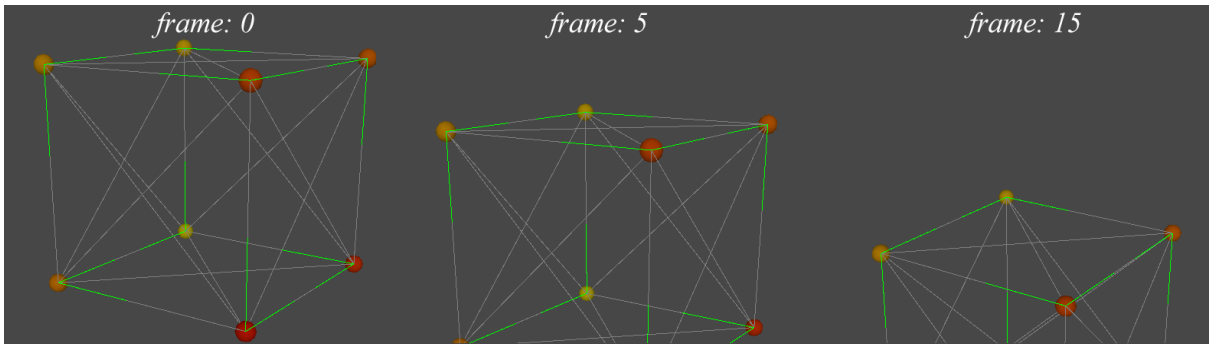
6.1.2.3 Visualització

La visualització del resultat és una de les qüestions més importants. D'aquesta manera podem comprovar si l'objectiu compleix els requeriments proposats. També ens pot ser útil en futures etapes per afrontar les necessitats de depuració de codi.

A l'altura d'aquesta iteració, encara no és un requeriment que es renderitzi una malla. Ho serà més endavant.

Partint d'aquesta premissa, la situació es resol temporalment renderitzant a través dels mètodes per dibuixar *Gizmos* de *Unity*. Es sobreescriuen les crides per dibuixar en pantalla, seguint la mateixa estructura de classes proposada. És a dir que una partícula és capaç de saber com es dibuixa a sí mateixa, i un *Softbody* sap com es dibuixa un sistema de partícules. Les crides en cascada s'originen de *Softbody* perquè és l'únic que hereta de *MonoBehaviour*.

Figura 11. Visualització del sistema de partícules



Font: Elaboració pròpia

El que veiem en aquest moment, són les partícules interconnectades que cauen infinitament cap avall, i no tenen cap impediment que les freni.

6.1.3 Valoració de la iteració

La valoració del primer cicle és positiva en tant que s'aplica estrictament la metodologia en espiral proposada, i el temps de la iteració supera l'esperat.

L'assoliment dels objectius és comprovable en les posteriors etapes, amb vídeo de la simulació final. També es pot observar el resultat de la iteració a la Figura 11. Arribat en aquest punt, la metodologia en espiral permet planejar la següent iteració.

6.2 Segona iteració

6.2.1 Objectius de la iteració

La segona iteració es basa en aplicar els *Constraints* que s'han exposat al marc teòric. Concretament el *Mass-Spring Constraint*, el qual permetrà simular l'elasticitat de teixits.

L'objectiu de la iteració és l'extensió del sistema de partícules actual perquè sigui capaç de resoldre $j \in [1, \ldots, M]$ restriccions. Al final de la iteració s'obté un sistema de partícules on hi ha interacció entre partícules connectades. Es pot comprovar amb el vídeo de la simulació final si s'ha complert l'objectiu.

6.2.2 Desenvolupament de la iteració

6.2.2.1 Arquitectura

Una de les solucions proposades al marc teòric, afronta els *Constraints* amb una estructura d'objectes generalitzats. Aquesta solució permet iterar sobre tots els *Constraints* i projectar-los en un moment determinat del bucle de simulació, tractant-los tots per igual. Ja que hereten del mateix objecte pare.

La solució que s'aplica en aquest treball, no és una generalització tant extensa perquè no es preveu incloure més *Constraints*. Si en un futur fos desitjat, es podria crear un sistema de classes que ho permeti, i tingui en compte els diferents complexes geomètrics.

6.2.2.2 Mass-Spring Constraint

Una de les restriccions nuclears del sistema de partícules, és el lligam que té una partícula amb les seves connexions. *Mass-Spring Constraint,* també anomenat

restricció de distància, pot programar-se de diverses formes. Com que aquest treball es focalitza en PBD, la solució és modificar únicament les posicions de les partícules, sense interferir en la seva velocitat implícita.

S'aplica *Gauss-Seidel* per projectar el conjunt de *Constraints*. Aquest fet es deu a la naturalesa seqüencial de Unity. Tanmateix, també podria projectar-se amb mètodes de *Jacobi* tal com s'ha vist al marc teòric.

Finalment, la implementació de la restricció de distància esdevé un tros de codi *branchless* que simula el comportament d'una molla. S'utilitzen les fórmules exposades a la secció de teoria. El fet que sigui *branchless* simplifica el flux de lògica, lo qual és una petita optimització que només és apreciable i té impacte a gran escala.

6.2.2.3 Fixed Constrain

Per comprovar el que el sistema funciona, hi ha un altre fenomen que és important a l'altura d'aquesta iteració. És l'ús d'un altre restricció, que serveix per fixar la posició d'una partícula en una coordenada concreta.

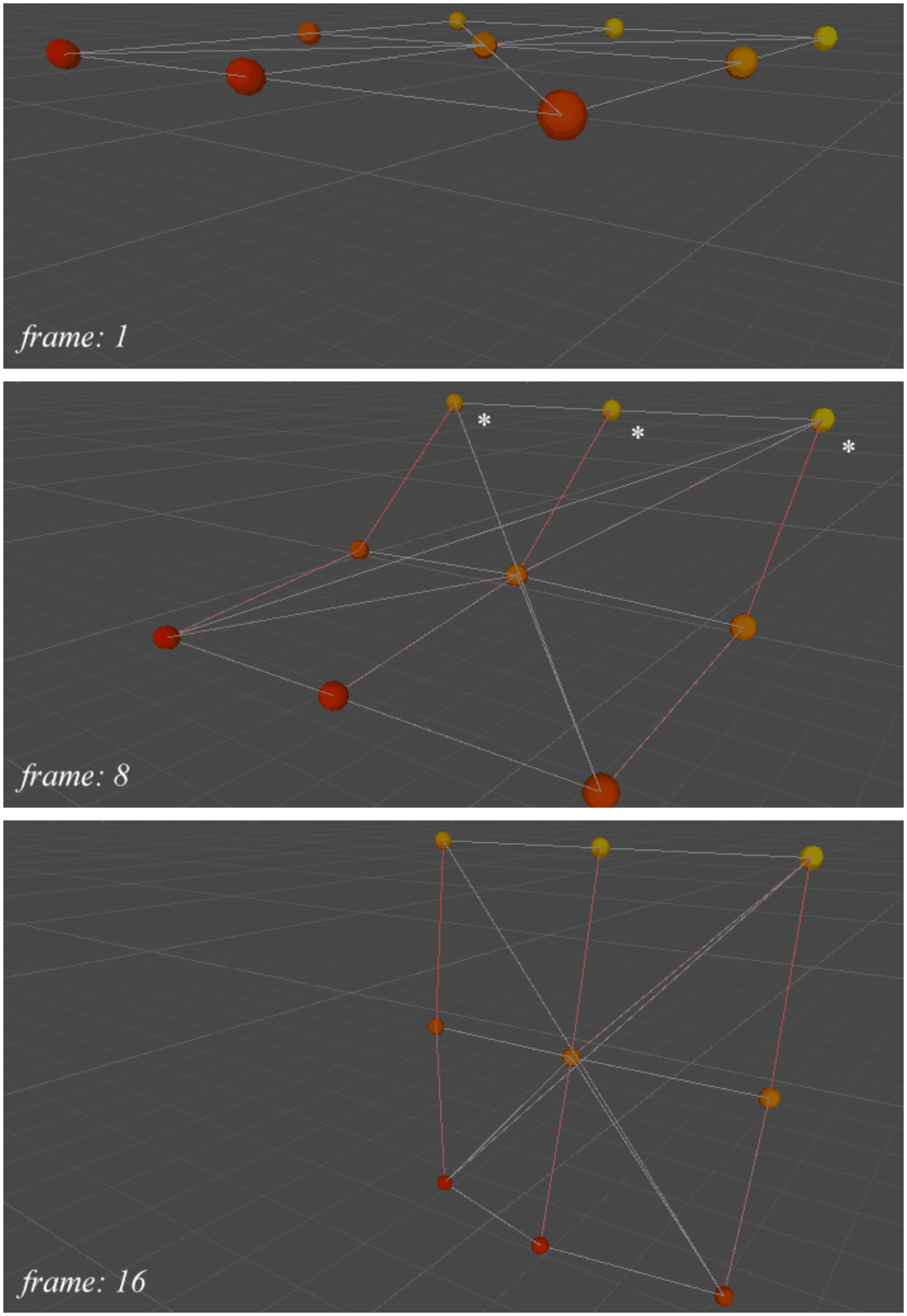
Aquesta restricció és importat que sigui la última de totes en el bucle de simulació. És a dir, que l'ordre importa a l'hora de projectar.

El resultat visual que s'obté simula un tros de roba en un estenedor, sostingut per una pinça. Alhora es pot visualitzar l'elasticitat del teixit format entre partícules.

A la Figura 12, en color vermell es pot visualitzar el sobre-estirament de les connexions. Quan el color de la connexió està en repòs, es visualitza en gris.

Les partícules marcades en asteriscs (*) estan afectades pel *fixed constraint*, de forma que no es mouen amb el pas del temps, i ignoren les lleis de la física. La resta de partícules estan afectades per la gravetat.

Figura 12. Simulació de teixits amb Mass-Spring Constraint



Font: Elaboració pròpia

6.2.3 Valoració de la iteració

La valoració de la iteració és positiva en tant que s'assoleixen els objectius. Per altre banda, s'ha anticipat la finalització de la iteració una setmana. Això vol dir que el plantejament original preveia una setmana més. Aquest fet permet començar la tercera iteració amb més calma, sense necessitat de desfasar la resta de desenvolupament.

Davant d'aquestes circumstàncies, la metodologia en espiral permet planejar la següent iteració.

6.3 Tercera iteració

6.3.1 Objectius de la iteració

L'objectiu de la tercera iteració és vincular una malla i sotmetre-la a les forces de les partícules. Al final de la iteració s'obté un sistema de *Skinning* amb el mètode LBS. Per fer-ho s'apliquen alguns dels principis que es detallen al marc teòric. El requeriment de la Taula 1 és comprovable si es pot visualitzar una malla afectada per les partícules en moviment.

6.3.2 Desenvolupament de la iteració

6.3.2.1 Recursos

Prèviament al desenvolupament, es necessita una font de recursos per fer comprovacions amb objectes 3D. La intenció és obtenir dos nivells de complexitat que representin un mateix objecte. Una versió *High Poly* i una versió reduïda de la mateixa representada en un graf. No es desitja utilitzar recursos provinents de tercers per qüestions legals i/o tècniques; evitar drets de còpia i adequar el model pel cas d'ús.

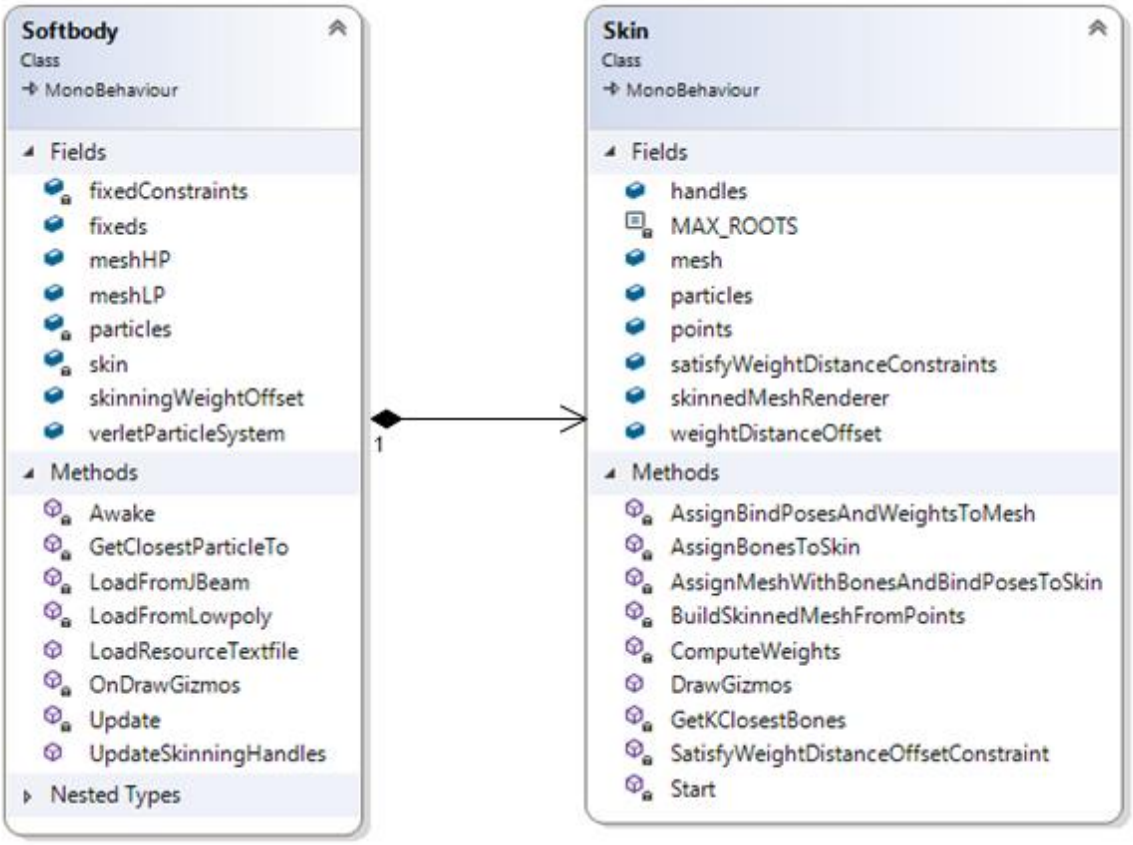
El recurs *High Poly* es modela amb eines de codi obert, concretament a través de *Blender*. La versió reduïda té dos mètodes d'importació. El primer mètode és la generació d'un graf a partir d'una malla. El segon mètode és la lectura d'un fitxer codificat amb un graf directament.

6.3.2.2 Arquitectura

El disseny de classes a l'hora d'implementar *Skinning* es veu poc afectat en quant a nous objectes, ja que només s'introdueix la classe *Skin*. Malgrat, s'introdueixen nous camps i mètodes per la resta d'objectes existents. A continuació es mostra el diagrama de classes de la classe *Skin* i el seu acoblament a la resta del sistema.

La classe *Skin* només es relaciona amb el *Softbody* per agregació. El *Softbody* actua com a controlador i afegeix el component de *Skin* al *GameObject* quan s'inicialitza. Seguidament en cascada, el component de *Skin* afegeix un altre component de Unity per visualitzar la malla, el *SkinnedMeshRenderer*. El darrer component, internament aplica *Linear Blend Skinning* (LBS).

Figura 13. Diagrama de classes amb incorporació de Skin



Font: Elaboració pròpia

Abans de poder utilitzar el mecanisme d'animació LBS, s'ha d'especificar com es vincula la malla amb les partícules del sistema. Aquest procediment s'anomena *Binding*. El *Binding* executa un algorisme per crear ossos arrelats a un nombre fix de partícules, que alhora afecten a un conjunt de vèrtex. El nombre fix es deu a una restricció imposada per Unity que només permet ossos de màxim quatre vèrtex.

La crida del mètode *Skin::BuildSkinnedMeshFromPoints* inicia el procés manual de *Binding*. El procés consta d'un conjunt d'etapes que s'expliquen a continuació. Primerament l'algorisme recorre les partícules del sistema. Per cada partícula es crea un sistema de coordenades en una matriu 4x4, el qual anomenem *bindPoses*. A Unity, els ossos han d'existir a l'escena, per tant és necessari crear nous *GameObjects* per representar ossos anomenats *Handles*.

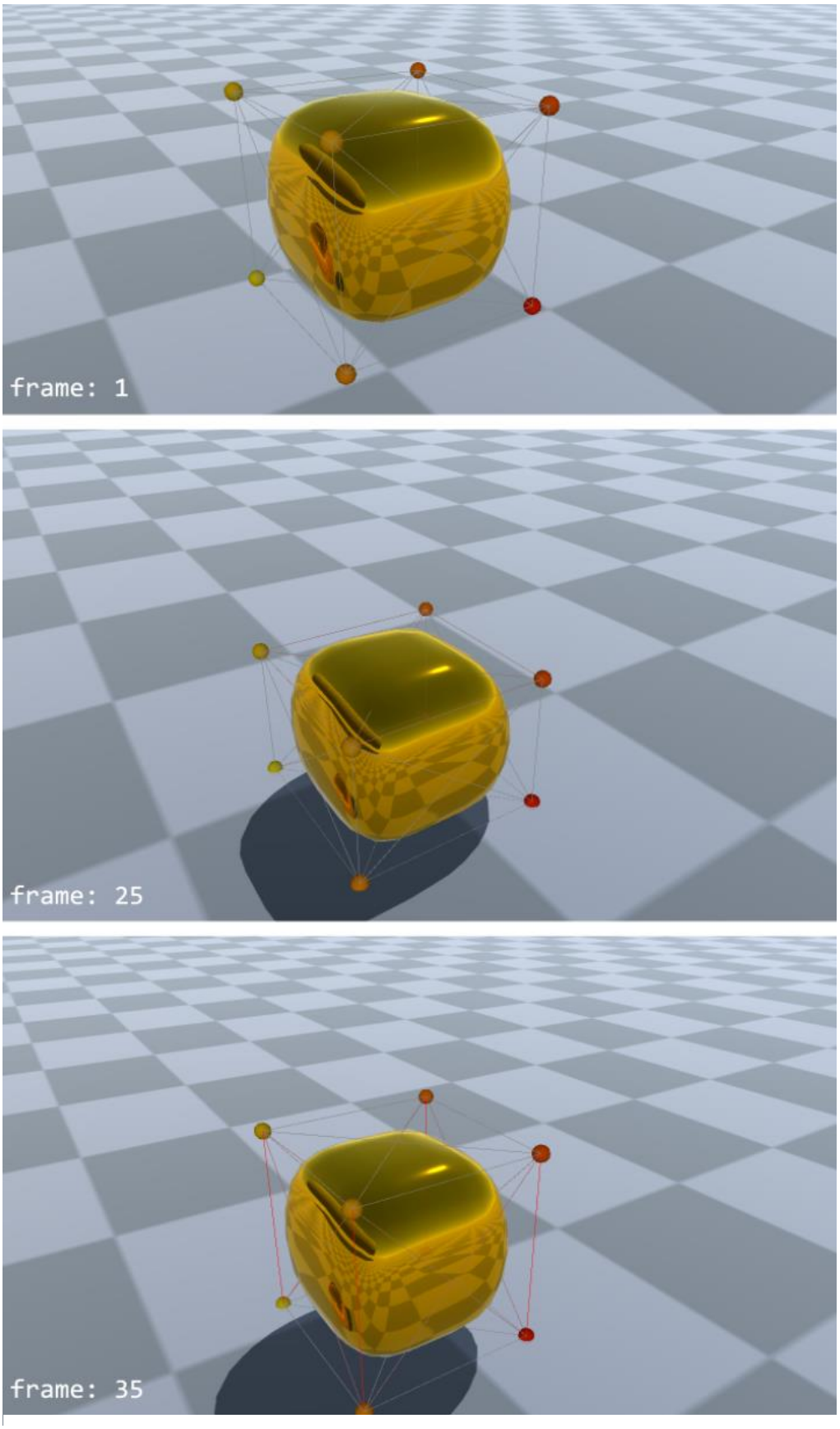
A continuació es recorren tots els vèrtex de la malla *High Poly* i es realitza *K-Nearest-Neighbours* (KNN), on *K* representa el màxim nombre d'arrels. Tot seguit es computa la distribució de pesos per saber la influència de cada os.

Finalment s'assigna el *Binding* que hem realitzat a la malla, format pels pesos i les *bindPoses*. També s'assignen els ossos al *SkinnedMeshRenderer*. Per últim es vincula la malla al *SkinnedMeshRenderer*.

Per tal d'actualitzar les *Handles* s'usa el mètode *Softbody::UpdateSkinningHandles* una vegada per cada fotograma. L'actualització de *Handles* és independent del nombre d'iteracions que comporta la simulació. Un *Handle* és determinat per una transformació composta d'una translació i una rotació. La translació ve determinada per la posició actual de les partícules. La rotació s'interpreta a través de les connexions entre partícules, pel qual és necessari extreure la part rotacional d'una matriu 3x3 tal com s'ha explicat al marc teòric.

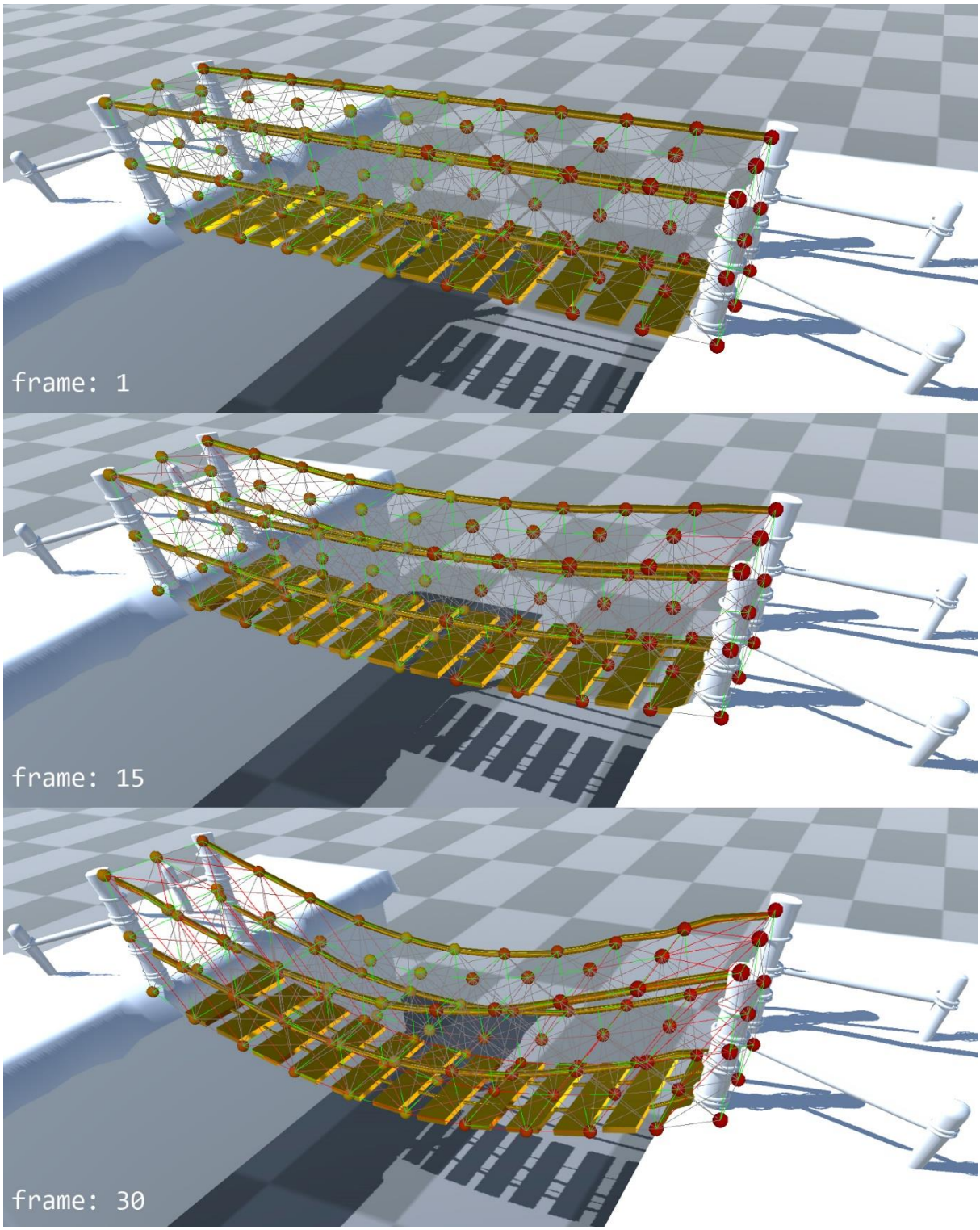
A continuació es mostra un exemple del resultat obtingut al final de la iteració. Es pot observar que el cub està format per molts polígons per representar una superfície corba. El cub és governat per un sistema de partícules simplificat que respon a les lleis de la física, representat amb esferes vermelles. Al fotograma número 25 es pot apreciar l'aplanament provocat per l'impacte i la reacció del cos. Veure Figura 14 i 15.

Figura 14. Simulació amb Skinning de malles



Font: Elaboració pròpia amb Unity

Figura 15. Simulació amb Skinning de malles complexa



Font: Elaboració pròpia amb Unity

Un altre requeriment desenvolupat en aquesta iteració, és la restricció de col·lisió amb altres objectes. La restricció de col·lisió es satisfà de forma unilateral com s'ha exposat a la secció teòrica amb les seves formulacions.

Unity fa ús del motor de *Nvidia* anomenat *PhysX* per resoldre col·lisions entre cossos rígids. *PhysX* és un motor de físiques consolidat que no té per objectiu suportar cossos tous en el seu horitzó temporal. Dins la gama de productes de *Nvidia* es destaca el motor *FleX,* orientat a cossos tous i simulacions de fluids.

Degut a l'entorn escollit, amb les sabudes conseqüències, no hi ha forma de reaprofitar el motor de físiques de Unity per resoldre les col·lisions entre partícula i *Game Object*. El repte implica anar en contra del propi motor perquè s'ha d'implementar una alternativa a la solució nativa.

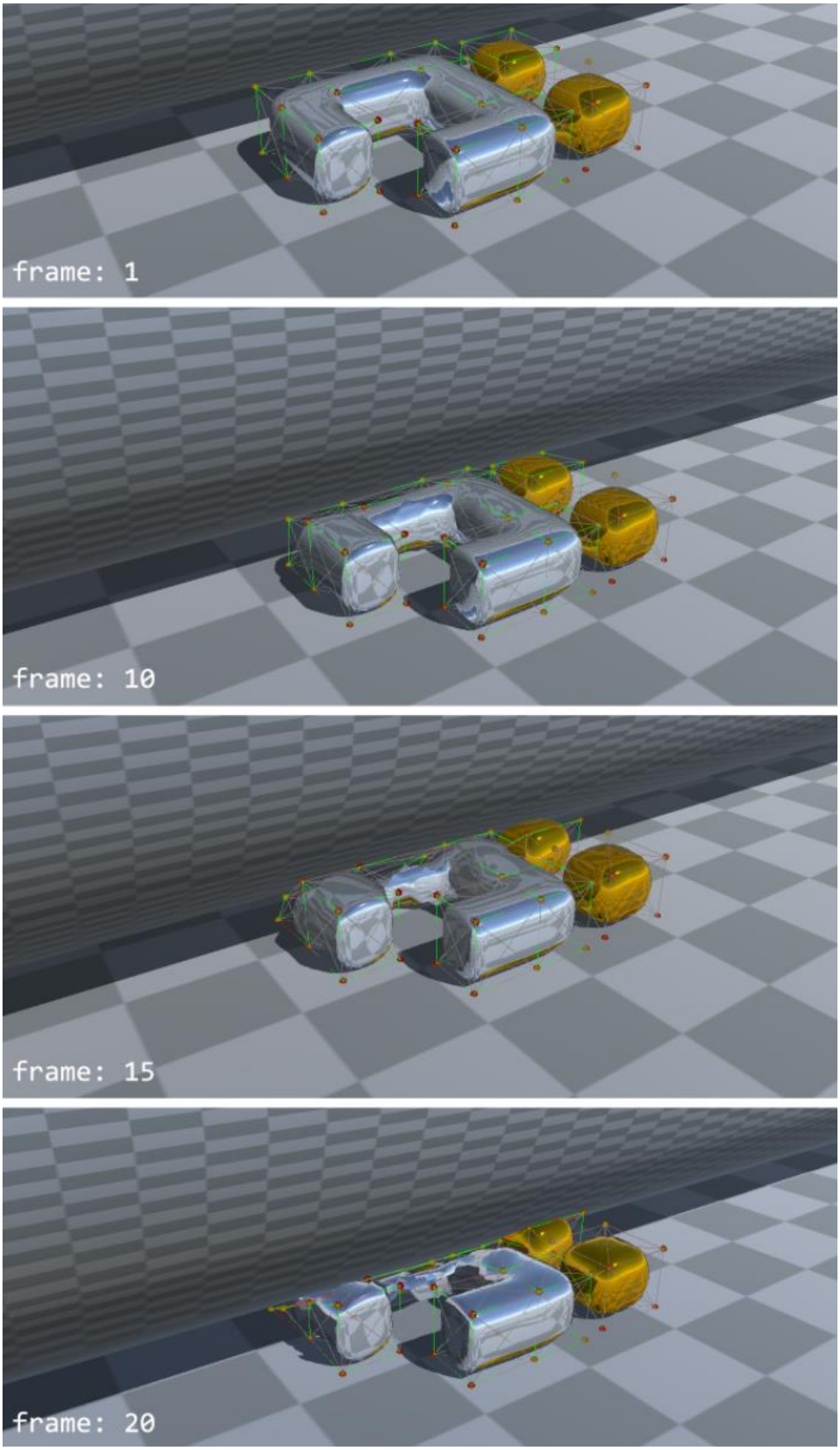
És desitjat reaprofitar les definicions de geometria dels *Colliders* de Unity. D'aquesta manera les partícules i els *Game Objects* poden interaccionar entre si, sempre i quan tinguin un *Collider* assignat. Finalment s'obté un sistema de detecció-resposta de col·lisions unidireccional. El sistema és unidireccional perquè les partícules no ofereixen cap resistència física cap als *Rigid Bodies*, però els *Rigid Bodies* sí que desplacen les partícules.

La detecció de col·lisió es resol a través d'utilitats de *Unity* per saber si un punt és contingut en un *Collider*. La resposta de col·lisió és un sistema *penalty-based* que posiciona la partícula a la superfície de nou per on ha penetrat. Per determinar la nova posició s'ha programat un mètode genèric per qualsevol *Collider* que llança un raig invertit des de dins de la malla cap en fora. Finalment es projecta la nova posició de la partícula satisfent el *Constraint*.

Al tractar-se d'un mètode genèric per resoldre col·lisions, s'ha pogut comprovar petites imprecisions en els extrems d'alguns tipus de *Colliders*. Malgrat, no suposa un problema pel cas d'ús, ja que ens serveix per simular un terra sòlid.

A la Figura 16 es pot observar la simulació d'un cos rígid en moviment que aixafa el cos tou. El cos tou és format per diversos elements que representen el logotip del Tecnocampus.

Figura 16. Simulació de múltiples objectes amb logotip del Tecnocampus



Font: Elaboració pròpia amb Unity

6.3.3 Valoració de la iteració

La retrospectiva de la iteració és positiva perquè s'ha demostrat assolir els objectius. A pesar d'haver assolit els objectius, ens hem trobat amb diversos reptes que han dificultat el desenvolupament.

A continuació s'exposen alguns dels reptes tècnics que han requerit més temps del esperat. Un dels majors reptes ha estat determinar la rotació dels ossos a la simulació de partícules. Com s'ha exposat a la secció de teoria, es tracte d'un problema matemàtic es coneix com *Whaba Problem*. Inicialment s'ha intentat codificar la implementació proposada per Matthias Müller et al. (2016). La implementació realitzada ha generat errors subtils de moviment per causes desconegudes. Finalment s'ha implementat un altre mètode basat amb recursos de la classe *Quaternion* pròpia de Unity.

Un altre repte ha estat l'adaptació de les restriccions de *PhysX*, ja que Unity no és un motor dissenyat per simular cossos tous. A pesar de conèixer aquesta restricció prèviament al desenvolupament, s'ha procurat adaptar-se al màxim al motor. S'evita crear nous tipus d'objectes incompatibles amb les funcionalitats de Unity i es reaprofita la base dels *Colliders*.

Un altre aspecte a valorar, és la possible millora a la implementació del *Binding* de la malla amb els ossos. Hauria sigut millor construir una estructura de dades eficient per fer consultes KNN. No s'ha implementat una solució més eficient perquè no és objecte d'estudi en aquest treball. Tampoc s'ha contemplat perquè forma part d'un preprocés sense impacte en la eficiència de la simulació.

No obstant dels reptes i les possibles millores, no s'ha produït una dislocació del temps respecte el cronograma inicial. S'ha finalitzat el desenvolupament amb el temps previst amb èxit.

7 Resultats

7.1 Objectius

Aquest apartat exposa els resultats obtinguts del desenvolupament del treball. En primera instància cal destacar que s'ha implementat tots els requeriments desitjats des del inici, com es mostra a la Taula 1.

De la mateixa manera, no s'ha arribat a implementar la interacció per part de l'usuari amb els cossos tous. Tampoc s'ha implementat la col·lisió entre ambdós sistemes de partícules. Les causes han estat en primer lloc la falta de temps, i en segon lloc que no estaven contemplats des del inici. Malgrat, s'ha fet menció a aquests requeriments perquè són els futurs passos a seguir un cop finalitzi el treball. Si la planificació del cronograma hagués estat superada, haguessin estat les funcionalitats imminents.

Pel que fa a l'eina final, està lluny de ser usable a nivell de producció. A pesar de haver obtingut un sistema de partícules funcional, com a llibreria encara no està a punt. Falta polir i reiterar sobre la implementació per facilitar l'ús de cara l'usuari. En relació al codi desenvolupat, és sabuda la quantitat de canvis i refaccions possibles. Les refaccions pendents són per facilitar l'escalabilitat del codi i capacitar-lo amb nous efectes. També és sabuda la quantitat de operacions que poden ser optimitzades, tal com s'ha mencionat al llarg del desenvolupament.

7.2 Valoració final

A desgrat de les parts millorables, o pendents de implementar, es valora satisfactòriament el desenvolupament del treball. S'ha estat estrictament fidel al calendari en tot moment, veure Taula 2. Les úniques variacions de temps han estat de màxim un o dos dies per iteració. Per la qual cosa no s'ha fet un esment rellevant a la majoria d'iteracions.

A pesar que el desenvolupament s'ha cenyit al cronograma establert, s'ha detectat possibles millores en el calendari respecte la redacció. Com es preveia, la part de redacció només tenia lloc al inici del projecte. Malauradament la part de redacció ha

tingut cabuda fins al final del desenvolupament. S'hauria d'haver previst espais de redacció entre fites. D'aquesta manera s'hagués dedicat el temps a la implementació i redacció de forma concurrent, i no simultàniament amb el desordre que comporta. Afavorint la concentració per cada aspecte de manera independent. Així i tot, com s'ha explicat, no hi ha hagut grans desviacions en el calendari, però si que hi ha hagut més pressió de la esperada.

8 Referències

8.1 Bibliografia

Aburumman, N., & Fratarcangeli, M. (2015). Position-Based Skinning for Soft Articulated Characters. *Computer Graphics Forum*, *34*. https://doi.org/10.1111/cgf.12533

Aifantis, E. C. (1987). The physics of plastic deformation. *International Journal of Plasticity*, *3*(3), 211-247. https://doi.org/10.1016/0749-6419(87)90021-0

Boehm, B. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, *11*(4), 14-24. https://doi.org/10.1145/12944.12948

Boubaker, M. B., Haboussi, M., Ganghoffer, J.-F., & Aletti, P. (2009). Finite element simulation of interactions between pelvic organs: Predictive model of the prostate motion in the context of radiotherapy. *Journal of Biomechanics*, *42*(12), 1862-1868. https://doi.org/10.1016/j.jbiomech.2009.05.022

Bridson, R., Fedkiw, R., & Anderson, J. (2002). *Robust Treatment of Collisions, Contact and Friction for Cloth Animation*. 10.

Clough, R. W. (1960). *The Finite Element Method in Plane Stress Analysis*. American Society of Civil Engineers.

Farrell, J. L., Stuelpnagel, J. C., Wessner, R. H., Velman, J. R., & Brook, J. E. (1966). A Least Squares Estimate of Satellite Attitude (Grace Wahba). *SIAM Review*, *8*(3), 384-386. https://doi.org/10.1137/1008080

García, F. G., Paradinas, T., Coll, N., & Patow, G. (2013). *Cages: A multilevel, multi-cage-based system for mesh deformation. *ACM Transactions on Graphics*, *32*(3), 24:1-24:13. https://doi.org/10.1145/2487228.2487232

GDC Vault—Real-Time Deformation and Fracture—Finite Element Simulation and its Use in STAR WARS: THE FORCE UNLEASHED. (2009). Recuperat 30 gener 2021, de https://www.gdcvault.com/play/1279/Real-Time-Deformation-and-Fracture

Hrennikoff, A. P. (1940). *Plane stress and bending of plates by method of articulated framework* [Thesis, Massachusetts Institute of Technology]. https://dspace.mit.edu/handle/1721.1/64833

Jakobsen, T. (2001). Advanced character physics. *In Game Developers Conference Proceedings*.

JBeam Physics Theory—BeamNG. (2015). Recuperat 30 gener 2021, de https://wiki.beamng.com/JBeam_Physics_Theory

Macklin, M., Müller, M., & Chentanez, N. (2016). XPBD: Position-based simulation of compliant constrained dynamics. *Proceedings of the 9th International Conference on Motion in Games - MIG '16*, 49-54. https://doi.org/10.1145/2994258.2994272

Macklin, M., Storey, K., Lu, M., Terdiman, P., Chentanez, N., Jeschke, S., & Müller, M. (2019). Small steps in physics simulation. *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '19*, 1-7. https://doi.org/10.1145/3309486.3340247

Maggiorini, D., Ripamonti, L. A., & Sauro, F. (2014, maig 29). *Unifying Rigid and Soft Bodies Representation: The Sulfur Physics Engine* [Research Article]. International Journal of Computer Games Technology; Hindawi. https://doi.org/10.1155/2014/485019

McDonnell, K., & Qin, H. (2007). PB-FFD: A point-based technique for free-form deformation. *J. Graphics Tools*, *12*, 25-41. https://doi.org/10.1080/2151237X.2007.10129242

Mercier, J. P., Zambelli, G., & Kurz, W. (2002). Chapter 12—Factors influencing mechanical properties. En J. P. Mercier, G. Zambelli, & W. Kurz (Ed.), *Introduction to Materials Science* (p. 279-320). Elsevier. https://doi.org/10.1016/B978-2-84299-286-6.50018-4

Michaud, L. (2016). *State of Video Game Industry Trends, analyses and in-depth market outlook Digital Entertainment About IDATE DigiWorld DigiWorld Research Programme*.

Milaszewicz, J. P. (1987). Improving Jacobi and Gauss-Seidel Iterations. *Linear Algebra and Its Applications*, *93*, 161-170. https://doi.org/10.1016/S0024-3795(87)90321-1

Müller, M. (2008). Hierarchical Position Based Dynamics. *VRIPHYS*. https://doi.org/10.2312/PE/vriphys/vriphys08/001-010

Müller, Matthias, Bender, J., Chentanez, N., & Macklin, M. (2016). A robust method to extract the rotational part of deformations. *Proceedings of the 9th International Conference on Motion in Games*, 55-60. https://doi.org/10.1145/2994258.2994269

Stam, J. (2009). *Nucleus: Towards a Unified Dynamics Solver for Computer Graphics*. 1-11. https://doi.org/10.1109/CADCG.2009.5246818

The Most Impressive Physics Engine You've Never Seen—IGN. (2012). Recuperat 31 gener 2021, de https://www.ign.com/articles/2012/10/01/the-most-impressive-physics-engine-youve-never-seen

Tsai, T.-C. (2017). Position Based Dynamics. En N. Lee (Ed.), *Encyclopedia of Computer Graphics and Games* (p. 1-5). Springer International Publishing. https://doi.org/10.1007/978-3-319-08234-9_92-1

van Ratingen, M., Williams, A., Lie, A., Seeck, A., Castaing, P., Kolke, R., Adriaenssens, G., & Miller, A. (2016). The European New Car Assessment Programme: A historical review. *Chinese Journal of Traumatology*, *19*(2), 63-69. https://doi.org/10.1016/j.cjtee.2015.11.016

Verlet, L. (1967). Computer «Experiments» on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, *159*(1), 98-103. https://doi.org/10.1103/PhysRev.159.98

Xu, L., Lu, Y., & Liu, Q. (2018). Integrating viscoelastic mass spring dampers into position-based dynamics to simulate soft tissue deformation in real time. *Royal Society Open Science*, *5*, 171587. https://doi.org/10.1098/rsos.171587

Zibrek, K., Martin, S., & McDonnell, R. (2019). Is Photorealism Important for Perception of Expressive Virtual Humans in Virtual Reality? *ACM Transactions on Applied Perception*, *16*(3), 14:1-14:19. https://doi.org/10.1145/3349609

8.2 Ludografia

Krome Studios (2008). *Star Wars: The Force Unleashed* (Playstation 2) [Videojoc]. California: LucasArts.

BeamNG (2015). *BeamNG.drive* (PC) [Videojoc]. Bremen: BeamNG.

Rigs of Rods (2011). *Rigs of Rods* (PC) [Videojoc]. Ricordel, P. M., Fisher, T., & Ohlidal, P.

IO Interactive (2000). *Hitman: Codename 47* (PC) [Videojoc]. Londres: Eidos Interactive Ltd.

9 Annex

9.1 Codi Font

Ruta CD: Annexos\Producte\CodiFont.zip

9.2 Instruccions d’ús

Ruta CD: Annexos\Instruccions\Instruccions.pdf

9.3 Vídeo de la simulació final

Ruta CD: Annexos\Simulacio\Video.mp4