

# Índice general

[Índice general 2](#_Toc294693371Toc294478847)

[Introducción 4](#_Toc294693372Toc294478848)

[Motivación 4](#_Toc294693373Toc294478849)

[Descripción del problema 4](#_Toc294693374Toc294478850)

[Organización del documento 4](#_Toc294693375Toc294478851)

[Estado del arte 5](#_Toc294693376Toc294478852)

[Reconstrucción 3D 5](#_Toc294693377Toc294478853)

[Correspondencia- mapping 5](#_Toc294693378Toc294478854)

[Método de triangulación 7](#_Toc294693379Toc294478855)

[Stero visión 8](#_Toc294693380Toc294478856)

[Structured light 9](#_Toc294693381Toc294478857)

[Calibración 9](#_Toc294693382Toc294478858)

[Procesamiento de nube de puntos 9](#_Toc294693383Toc294478859)

[Aplicaciones para *video mapping* existentes 10](#_Toc294693384Toc294478860)

[Modul8 10](#_Toc294693385Toc294478861)

[VMDX 12](#_Toc294693386Toc294478862)

[VVVV 14](#_Toc294693387Toc294478863)

[Aportes 14](#_Toc294693388Toc294478864)

[Solución planteada - VMT (The Video Mapping Tool) 14](#_Toc294693389Toc294478865)

[Objetivo 15](#_Toc294693390Toc294478866)

[Descripción general 15](#_Toc294693391Toc294478867)

[Descripción de la arquitectura 16](#_Toc294693392Toc294478868)

[Componentes 16](#_Toc294693393Toc294478869)

[Clases 17](#_Toc294693394Toc294478870)

[Colaboración 17](#_Toc294693395Toc294478871)

[Descripción de módulos 20](#_Toc294693396Toc294478872)

[Motor gráfico: *vmt\_engine* 20](#_Toc294693397Toc294478873)

[Interfaz de usuario 21](#_Toc294693398Toc294478874)

[Multi proyector 25](#_Toc294693399Toc294478875)

[Tratamiento de malla 25](#_Toc294693400Toc294478876)

[Calibración 25](#_Toc294693401Toc294478877)

[Efectos 25](#_Toc294693402Toc294478878)

[Conclusiones 25](#_Toc294693403Toc294478879)

[Dificultades encontradas, limitaciones y posibles mejoras 26](#_Toc294693404Toc294478880)

[Adquisición 26](#_Toc294693405Toc294478881)

[Calibración 26](#_Toc294693406Toc294478882)

[Conclusiones de la autoevaluación 26](#_Toc294693407Toc294478883)

[Prueba de desempeño 26](#_Toc294693408Toc294478884)

[Conclusiones 26](#_Toc294693409Toc294478885)

[ANEXO I – Eventos realizados 26](#_Toc294693410Toc294478886)

[ANEXO II - Casos de uso relevantes 26](#_Toc294693411Toc294478887)

[Glosario 27](#_Toc294693412Toc294478888)

[Bibliografía 27](#_Toc294693413Toc294478889)

[Índice de figuras 27](#_Toc294693414)

[Índice de cuadros 27](#_Toc294693415)

# Introducción

*Video mapping* es una técnica utilizada en tecnología audiovisual que logra transformar, crear, expandir, amplificar e interpretar espacios físicos.

Software especializado es utilizado para deformar y enmascarar las imágenes y videos para que se ajusten a las formas irregulares de la superficie.

## Motivación

Realizar una herramienta que permita manejo de modelos tridimensionales, sobre los cuales poder diseñar un show de video-mapping.

Permitir utilizar las deformaciones propias de las superficies como efectos en los diseños de artistas Vj’s generando visualización del show en un espacio virtual que simule la escena tridimensional.

## Descripción del problema

El presente proyecto consiste en un estudio del estado del arte de las técnicas de *video mapping* y modelado automático de geometrías, para luego implementar un producto que integre modelado de la escena 3D, mapeo de imágenes o videos y definición de efectos sobre objetos de la escena mediante un editor 3D finalmente proyección del diseño especificado.

## Organización del documento

2- Introducción al problema de correspondencia, modelado automático de superficies tridimensionales, técnicas de construcción y depuración de modelos tridimensionales, características principales de herramientas existentes para realización de *video mapping*.

3- Descripción de la solución.

4- Discusión de las técnicas y herramientas seleccionadas para utilizar en el proyecto.

# Estado del arte

En el estudio del Estado del Arte se incluirán técnicas existentes de reconstrucción de mapas tridimensionales (3D), relevamiento de funcionalidades de las aplicaciones de software disponibles hoy en día para la realización de video mapping.

Como información adicional, se incluirán extractos de una entrevista que tuvimos con un artista del medio local y comentarios de artistas internacionales quienes aportaron técnicas y aplicaciones que utilizan en la actualidad y lo que esperarían de una nueva aplicación desarrollada específicamente para realizar video mapping.

## Reconstrucción 3D

En esta sección se enumeran y explican brevemente técnicas para obtención de geometrías.

Introducción a métodos para reconstrucción de una superficie irregular a partir de una nube de puntos.

- Bibliotecas para generación de modelos 3D y modificación de los mismos.

- Técnicas de video mapping.

### Correspondencia- mapping

### Método de triangulación

Uno de los problemas a resolver en *computer vision* es el “problema de correspondencia” consiste en encontrar la transformación que permita definir correspondencia de puntos de dos instancias físicamente distintas de una misma escena. [1]

Método de triangulación

Determina las coordenadas (x,y,z) de un punto utilizando la posición del punto obtenida en las perspectivas de dos proyecciones dadas.

Los centros de perspectiva y planos de proyección son conocidos. [2]

Escena con dos dimensiones (2D)

Z

P

X

L

β

𝝰

O

Ɣ

origen de luz

cámara

h

b

objeto

d

Este método tiene como objetivo calcular la distancia *d* de la cámara al punto P a partir de los datos: ángulos , y la distancia *b* entre el proyector y la cámara.

El ángulo y la distancia b son dados por la calibración de la escena.

El ángulo esta dado por la geometría de la proyección.

→

Las coordenadas cartesianas quedan determinadas por:

Escena con tres dimensiones (3D)

P

Z

X

𝝰

O

Ɣ

origen de luz

f

b

objeto

x

y

p=(x,y)

Y

Se asume Z=f , f plano en el cual se proyecta el punto P(), obteniendo como resultado de la proyección el punto p(x,y).

El centro óptico del proyector está situado en el eje X.

P = (x,y) ,

Los métodos que resuelven obtener la geometría de una escena tridimensional sin tener contacto físico se pueden clasificar como activos o pasivos. [3]

Los métodos pasivos son aquellos en los que no es necesario utilizar luz adicional a la luz ambiente.

### Stero visión

Es un método pasivo para obtener la estructura tridimensional de una escena, se utiliza el método de triangulación sin intervención de luz auxiliar, la correspondencia es establecida entre dos o más imágenes.

Los algoritmos que utilizan este método son clasificados basados en diferencias de la geometría de la imagen, distintas estrategias para resolver la correspondencia entre puntos y en las diferentes estructuras computacionales utilizadas. [4]

Se realiza un pre proceso de las imágenes para identificar las principales características que se utilizarán, luego se define qué tipo de correspondencia se utilizará.

La principal desventaja de este método es que en caso de oclusión, hay regiones que no tienen correspondencia en las dos imágenes por lo tanto no se puede establecer la correspondencia de puntos, finalmente no se puede resolver el problema de correspondencia para estas regiones.

Las variantes que influyen en la geometría de la imagen son, ejes ópticos paralelos o no, paradigma binocular o multiocular.

La correspondencia entre una o más vistas de la escena se puede realizar basada en distintas estrategias, *separate area* o *features*.

- *separate area,* basada en correlación del brillo, intensidad, se utilizan patrones de brillo aplicados a un pixel y sus vecinos (utiliza principio de localidad), diferencias en la perspectiva de la imagen o cambios en luminosidad absoluta de la escena puede generar errores.

- *features,* las características usadas para la correspondencia son aristas, puntos o segmentos dadas por cambios de intensidad de la imagen, esta estrategia es más estable ante variedad de luminosidad absoluta y en la práctica correspondencia es más rápida.

La geometría convencional *stereo* tiene un par de cámaras y los ejes ópticos son paralelos entre sí, y perpendiculares a la línea base (dada por los dos centros ópticos de las cámaras).



### Structured light

[5]

*Structured light* es un método que resuelve el “problema de correspondencia”, el procedimiento consta en proyectar distintos patrones sobre la superficie a modelar, luego se realizan capturas de las proyecciones, analizando las deformaciones de los patrones proyectados se obtiene información de la posición (coordenadas tridimensionales), orientación y textura de la superficie.

Los patrones a utilizar en este método pueden ser variados se clasifican en cuatro tipos punto (*singled scanned dot*), línea (*slit line*), grilla (*grid*) y matriz de puntos (*dot matrix*).

Dependiendo del tipo de objeto y superficie a escanear se pueden tener problemas de oclusión, baja reflexión, puntos reflejados fuera del alcance de la cámara, como consecuencia hay pérdida de puntos proyectados que no tienen proyección en el plano imagen.

Estos problemas se pueden solucionar utilizando patrones codificados adecuados según el problema a resolver a continuación se listan tres grupos de patrones clasificados según dependencia temporal, propiedades de la luz proyectada y discontinuidad de profundidad de la superficie proyectada.

Dependencia temporal:

Estática- el patrón es limitado para escenas estáticas, son necesarias proyecciones de varios patrones distintos.

El movimiento de cualquier objeto de la escena mientras se realiza la obtención de los patrones proyectados producirá un error de correspondencia.

Dinámica- los objetos en la escena se pueden mover, se utiliza un único patrón de proyección.

Propiedades de la luz proyectada:

Binaria- cada uno de los puntos del patrón tiene dos posibles valores codificados con 0 y 1 respectivamente. Este valor representa opacidad y transparencia, ausencia o presencia de la luz proyectada en el objeto.

Escala de grises- cada punto del patrón tiene asociado un valor de gris que representa el nivel de trasparencia (o nivel de opacidad) del punto para la luz proyectada. Son necesarios dos pasos, primero se obtiene una imagen de la escena iluminada con la misma luz (sin variar la intensidad), luego se obtiene la referencia de luz necesaria para cancelar el efecto de reflejo de la superficie (depende directamente del tipo de superficie).La necesidad de estos dos pasos contribuyen a que este patrón también sea clasificado como estático.

Color- cada punto del patrón es asociado con un valor de tono. Los tonos deben ser bien diferenciados para alcanzar una segmentación eficiente.

Este tipo de patrones son limitados por el color de la escena, si presenta objetos de colores altamente saturados se producen pérdidas de regiones en el paso de segmentación que luego provoca errores en la decodificación.

Discontinuidad en profundidad de la superficie proyectada:

Periódica- la codificación se repite periódicamente a lo largo del patrón.

Esta técnica se utiliza para reducir el número de bits que codifican el patrón,  
como limitante la profundidad del objeto no puede ser mayor que la mitad de la longitud del período.

Absoluta- cada columna o fila del patrón proyectado tiene una única codificación, no sufre dependencia de discontinuidad de profundidad.

### Calibración

#### Calibración cámara

#### Calibración cámara-proyector

### Procesamiento de nube de puntos

## Aplicaciones para *video mapping* existentes

### Modul8

<http://www.modul8.ch/>

Es una aplicación diseñada para composición *y video-mixing* en tiempo real.

Una de las interfaces disponibles es una consola para manejar los efectos en tiempo real permitiendo complementar el diseño básico del *show* con intervenciones del vj realizadas en tiempo real.

La interfaz está estructurada en capas, en cada capa se puede agrupar efectos (manejo de audio y video) con especificaciones de los objetos particulares para la capa, luego la ejecución se realiza dependiendo del modo seleccionado, las opciones son: la capa activa, por capa y por conjunto de capas, las modificaciones realizadas se pueden ver de inmediato en la composición.

Los modelos visualizados sobre los que se diseña el *show* son representaciones bidimensionales de la escena (proyección 2D de la escena).

La arquitectura de la herramienta es modular admitiendo extensión de módulos implementados en python.

Modul8 es un software propietario disponible solo para plataforma MAC OS.

### VMDX

<http://vidvox.net/>

<http://vidvox.net/wiki/index.php/VDMX>

Es una herramienta que permite diseño y procesamiento de efectos en audio y video en tiempo real.

Está estructurado por capas, cada capa tiene una fuete que define básicamente como lucirá, luego es modificada por los efectos aplican a la capa, la imagen resultante se unirá con la imagen resultante de las otras capas (este procedimiento se realiza en un orden dado por la definición de capas).

El gestor de capas es la interfaz disponible para crear, eliminar, agrupar y ordenar capas, luego cada capa puede ser editada, o diseñada por una interfaz particular (*layer window*) en la cual se define que tipo de capa es (posibles tipos son: película , quartz composer, variable según la fuente), que controles contiene, y efectos visuales a aplicarle.

Los distintos controles tienen sus propios editores, los cuales permiten su configuración.

El gestor de *plugins* permite visualizar los *plugins* existentes o crear nuevos, (*plugins* como módulos que permiten extender la funcionalidad).

VDMX es un software propietario disponible solo para plataforma MAC OS.

### VVVV

<http://vvvv.org/>

Es un entorno de programación gráfica para desarrollo y edición.

Tiene interfaces físicas que permiten modificación de gráficos en movimiento en tiempo real, permite la integración de audio, video y animaciones 3d.

Se maneja con un único modo para construir y ejecutar un programa, modo de ejecución (*runtime*), está continuamente calculando y procesando gráficos mientas que se está desarrollando.

Los proyectos se realizan utilizando una interfaz gráfica, definiéndose las operaciones a realizar como nodos, los *links* unen los nodos definiendo en conjunto un flujo de trabajo que representa las acciones o efectos a realizar durante el *show*.

Utiliza arquitectura cliente/servidor permitiendo generar un *show* con varios proyectores, en el proyecto se especifica para cada nodo donde se ejecutará (uno de los clientes o en el servidor).

La arquitectura es modular, provee de una interface basada en COM que permite incorporación de nodos (pueden estar escritos en c++, c#, delphi etc.)

VVVV es un software propietario (existiendo una versión sin costo para fines no comerciales). Es requerida plataforma MAC o PC.

VPT videoprojectiontool

<http://hcgilje.wordpress.com/VPT/>

<http://nervousvision.com/vpt/videoprojectiontool_v3.1_manual.pdf>

VPT es una herramienta para proyección en tiempo real, aplicaciones en las que se puede utilizar esta herramienta son: para proyección de videos en formas complejas (con efectos sobre los mismos), adaptar proyecciones a superficies, combinar diseños realizados con intervenciones en vivo, proyección sobre múltiples pantallas.

Se compone de fuentes, capas, mezcladores y controladores.

Las fuentes pueden ser: 8 películas QuickTime, 1 fuente sólida (útil para colorear una capa entera), 1 fuente distorsionada (para generar texturas distorsionadas), 1 fuente de texto, 2 *live*, 1 *buffer.*

El mezclador, mezcla dos fuentes (las fuentes disponibles para ser mezcladas son películas, *lives* y *buffer*).

A una capa se le puede aplicar deformaciones, escalado o modificación de posición en la pantalla de forma independiente (como máximo se definen 16 capas).

Dispone de un editor de mascaras permitiendo la creación máscaras propias.

Una nueva funcionalidad es permitir aplicar una maya a una capa, esto permite realizar modificaciones tridimensionales al diseño, (efecto que no está disponible en las herramientas similares)<http://hcgilje.wordpress.com/2011/05/26/vpt-5-5-preview/>

Los controladores son módulos realizados para soportar OSC, MIDI etc.

Esta herramienta se puede extender incorporando nuevas fuentes.

Es una herramienta sin costo, y opera sobre Mac o Windows.

Otras herramientas útiles o complementarias para realizar video mapping son:

Arkaos, 3D Studio max, After effects, Blender, Quartz Composer, Flash Catalyst.

<http://www.arkaos.net/>

<http://usa.autodesk.com/3ds-max/>

<http://www.adobe.com/es/products/aftereffects.html>

<http://www.blender.org/>

<http://developer.apple.com/graphicsimaging/quartzcomposer/>

<http://www.adobe.com/products/flashcatalyst.html>

## Aportes

En el siguiente punto describimos intercambios realizados con VJ’s e ingeniero, agradecemos Marcelo Vidal, Martin Borini, Viktor Vicsek y Kyle McDonald por sus aportes a la comprensión del problema.

Marcelo Vidal

<http://vimeo.com/vjchindogu>

En los trabajos de fachadas que ha realizado utiliza modelado en dos dimensiones.

Las herramientas más utilizadas para su diseño son:  
After effects, QuikTime, Photoshop, 3D studio, Modul8, VDMX, Resolume PC.

Con su experiencia considera que la técnica más eficiente que él ha logrado consta de obtener la geometría proyectada con el mismo proyector que luego utilizará en el show, fija la posición del mismo en el lugar que será el show y dibuja los contornos de la geometría (“a mano” sobre un software de modelado) sobre la superficie, obteniendo así el modelo en dos dimensiones de la superficie.

Luego realiza su diseño utilizando el modelo 2d obtenido, en el momento de realizar el show debe colocar el proyector en el lugar inicial que utilizó para obtener la geometría, y allí realiza algunos ajustes finales en la herramienta que ejecutará la proyección.

Nos indicó la dificultad existente para mover los proyectores que se utilizan, pueden pesar 200kg y no son posibles movimientos milimétricos para ajuste, por ello el considera que luego de armar la escena siempre será necesario un ajuste del modelo (software) a la superficie.

Una de las características importantes que remarcó es la posibilidad de modificar en tiempo real el show.

Las deformaciones que se producen al proyectar sobre una superficie irregular nos comentó que en el área de los VJ existen distintas ramas, una puede ser utilizar la deformación dada por la superficie junto con deformaciones de la imagen o video a proyectar para lograr un efecto en conjunto, y otra rama sería intentar modificar el video ó imagen para minimizar los efectos dados por la superficie irregular (se realizan efectos para anular las deformidades dadas por la superficie irregular), esta última opción nos indicó que en su opinión es la menos interesante en el ámbito artístico de los VJ’s.

Martin Borini

<http://vimeo.com/ailaviu>

<http://groups.google.com/group/clandestinaweekendnerd/web/taller-de-mapping>

Las principales inquietudes que nos comunica Martin son referentes al manejo tridimensional del problema, poder conocer la deformación que se dará en las superficies profundas para utilizarlo en el diseño del show, y correspondencia entre el modelado y la superficie, calibración de un modelo 3d.

En sus trabajos realizados sobre fachadas ha trabajado sobre fotos a nivel tomadas desde el mismo lugar donde se coloca el proyector, según su experiencia es mejor que trabajar sobre planos o medidas tomadas por él.

Otro punto en el cual expresó interés fue en casos en los que se utilizan más de un proyector, solucionar las “costuras” (casos bordes donde finaliza la proyección de un proyector y comienza la de otro), profundidad dada por la ubicación de cada proyector en el modelo (cada uno tendría su profundidad).

Viktor Vicsek

<http://vimeo.com/viktorvicsek>

Viktor ha implementado sus propios módulos de software para sus diseños, utiliza la técnica *Johnny Lee's automatioc proj calibration* (<http://johnnylee.net/projects/thesis/> ) con modificaciones propias para utilización de cámara sin sensores de luz.

Las herramientas que utiliza son VVVV para lograr los efectos de distorsiones, él construyó su propio *video* *timeline system* en adobe air para manejo de videos, y el proceso que comúnmente sigue es: tomar fotografías y obtener planos de la arquitectura, utiliza 3ds max para consturir el modelo 3d, luego realiza las mascaras utilizando este modelo, el mapping lo realiza con VVVV aplicando video o textura a las máscaras realizadas con *shaders.*

Para la utilización de múltiples proyectores aplica *edge blending* y *homography* a una máscara adicional.

Kyle McDonald

<http://www.kylemcdonald.net/>

Kyle nos comenta que actualmente está muy interesado en trabajar con sistemas *real time mapping,* por ejemplo utilizando sensores infrarrojos y proyectores permitiendo un *show* de *mapping* con escenas no estáticas (nos presentó como ejemplos un bailarín o un actor).

Nos confirma que la versión *phase-shift scanning* que él implementó (utilizando la técnica de *Zhang's three-phase*) tiene como requerimiento que la escena sea una figura continua, por ello una cara puede ser escaneada, pero no dos objetos simples como un cubo y una pirámide.

Él nos explica que *Zhang's three-phase* se basa en el principio de valores de profundidad propagados a través de la superficie, por lo tanto si dos superficies son discontinuadas no puede determinar las profundidades relativas a los dos objetos de la escena.

Ideas que nos sugirió:

One way is to use a "unit wavelength" -- i.e., the width (or height) of the projection. This means that phase (depth) information does not need to be propagated, and discontinuous positions can be determined relative to each other. If you have a projector and camera that are not gamma-calibrated, this will cause a very significant low-frequency "waviness" or "ripple" effect that makes the data difficult to use. I'm working on some automatic gamma calibration tools right now, but don't expect them to be done for a few months.  
  
Another approach is to use multiple frequencies. Start with a very low (unit wavelength) scan to determine the general depth. Then higher frequency scans to recover the more precise depth. There is also a hybrid approach developed recently using 5 fringe images where every image contains low and high frequency components.  
  
This leads me to an older technique, which is what "SL" originally referred to: gray code scanning. This is based on the idea of binary subdivision of a scene using gray codes. There's a demo here:  
  
<http://www.vimeo.com/3193063>  
  
But I haven't spent much time with it recently and there is a lot of room for improvement. For a better implementation, see:  
  
<http://mesh.brown.edu/byo3d/>  
  
Finally, if you don't need to acquire the mesh quickly, I recommend using the DAVID laser scanner system:  
  
<http://www.david-laserscanner.com/>  
  
Which uses a laser and webcam instead of a projector. It will give you higher quality data, but takes longer to scan a scene.

# Solución planteada - VMT (The Video Mapping Tool)

## Objetivo

Proveer de una herramienta que permita automatizar el mapeo de video sobre superficies irregulares, brindando al usuario la posibilidad de capturar una geometría, editar la misma para realizar posibles correcciones (ejemplo: reducir la nube de puntos en base a algoritmos disponibles), interfaz para diseño del *show* a partir del modelo 3D.

Finalmente se proveerá funcionalidad tanto para la generación de la proyección según lo definido como la posibilidad de reproducirla en vivo y realizar modificaciones en tiempo real.

## Descripción general

[Se describen las decisiones tomadas y cómo estas afectaron al alcance.

Se describe todo lo que hicimos y también las que no hicimos (obtención de nube de puntos, scripting, gráfico de música utilizado como interfaz para definir el momento de los eventos, selección de las caras de la maya en la gui)]

[

· La aplicación es distribuida, lo que permite tener varios proyectores en distintos nodos/equipos posiblemente alejados físicamente.

· Creación de un show desde una consola centralizada. Posibilidad de visualizar lo que se proyectaría en cada nodo.

· Edición del show interactiva (permite ver y testear lo que se está diseñando).

· Permite manipular objetos 2D y 3D

· Permite mapear imágenes y videos.

· Load/Save del show en formato XML estandard.

· Abstracción de zonas a proyectar mediante la creación de objetos mapeables.

· Calibración para cada nodo independiente resistente al movimiento de los proyectores.

· Mapeo sobre quads disjuntos útil entre otras cosas para solucionar problemas de costuras.

. Preprocesamiento de mallas, permitiéndose crear la misma a partir de una nube de puntos

· Creación de efectos y posibilidad de ajustarlos en forma individual.

· Lanzamiento de efectos de forma planificada por tiempo y por eventos de teclado

· Posibilidad de agregar audio al show.

· Manipulación de la linea del tiempo del show.

]

[

· Multiplataforma

· GUI basada en botoneras y lista de propiedades flotantes.

· Varios formatos de imágen y video

]

## Descripción de la arquitectura

### Componentes

- Procesamiento de la malla.

- Editor / server

- Nodo

- Engine

[diagrama de distribución de componentes]

### 

### Clases

Fig. X: Diagrama de clases de vmt\_engine

[Diagrama y descripción de clases del engine]

.- diagrama y descripción de clases de de VMTNode y VMT

.- diagrama y descripción de clases de la GUI

### Colaboración

. diagrama de send message y send message all para explicar comunicación consola/nodos

. ver de agregar alguno mas para explicar algún otro mecanismo

## Descripción de módulos

### Motor gráfico: *vmt\_engine*

Tanto la aplicación cliente como el servidor tienen como salida gráfica elementos bidimensionales y tridimensionales, ya sea en pantalla o proyectados.

Estos elementos son los que se utilizarán como base sobre los cuales se mapearán texturas estáticas como imágenes, o dinámicas como videos.

Una de las decisiones en el diseño de VMT fue la de separar la arquitectura en módulos reutilizables. El módulo *vmt\_engine* es utilizado tanto por la aplicación cliente como por el servidor. Este módulo es el encargado de representar los elementos gráficos y ofrece una interfaz para su manejo.

Los elementos bidimensionales son llamados *quads* y son básicamente cuadriláteros.

Estos quads se definen mediante las coordenadas bidimensionales de los cuatro vértices que los componen.

Una de sus utilidades es posicionarlos de forma tal que al ser proyectados cubran una superficie total o parcialmente. Luego sobre estos *quads* se aplicarán texturas, en forma de imagen o video, para que la proyección deforme estas texturas, adaptándolas a la forma del *quad*.

Los *quads* también pueden ser utilizados como máscaras para cubrir zonas de otros *quads* o elementos tridimensionales. Estas máscaras se utilizan con la finalidad de bloquear las partes, que por la geometría de la escena, podrían estar quedando mal proyectadas e iluminando zonas indeseadas. Una máscara es básicamente un *quad* opaco de color negro, ya que el color negro es el menos visible al ser proyectado.

Los otros elementos antes mencionados son los elementos tridimensionales y representan objetos más complejos y con mayor versatilidad para mapear texturas sobre ellos.

Estos objetos se cargan a partir de modelos en formato 3DS[The Labs: 3DS File Format, by Jeff Lewis - www.the-labs.com/Blender/3dsspec.html] y permiten cualquier geometría permitida por dicho formato.

La edición de la geometría de estos objetos es en general un tema complejo que lo resuelven bien varias aplicaciones de edición 3D como pueden ser 3DS Max[usa.autodesk.com/3ds-max] y Blender[www.blender.org] entre otros.

Estos programas, además de permitir editar la geometría manipulando vértices y caras, permiten definir materiales para aplicar propiedades comunes a una cierta selección de caras de la malla 3D [glosario]. VMT utiliza estas selecciones de caras para mapear texturas sobre ellas resolviendo de esta forma el mapeo en tres dimensiones.

El mapeo tridimensional se realiza básicamente definiendo las coordenadas de mapeo en cada vértice del modelo 3DS. Las aplicaciones de edición 3D antes mencionadas proveen herramientas para hacer esta tarea de forma muy intuitiva. Una utilidad de esto, por ejemplo, es la de definir las coordenadas de mapeo de un cilindro para mapear una textura en una columna cilíndrica.

Los elementos gráficos por sí mismos no son representables. Es necesario asignarles primero un material dentro de VMT y es a este material que se le asigna un color o una textura con la cual se representará el objeto.

En la implementación de los materiales se utilizan *shaders* [glosario] básicos que permiten combinar una textura con un color para lograr distintos resultados visibles.

Este color tiene un canal alfa [glosario] y por lo tanto se pueden lograr texturas y colores con niveles de transparencia. Estas texturas admitidas por los materiales pueden ser imágenes o videos.

Para visualizar los elementos tridimensionales es necesario definir un punto de vista. Este punto de vista junto con otras propiedades de visualización son asignadas a una cámara.

Los proyectores con los que se realiza el *show* de *video mapping* son representados con cámaras en vmt\_engine, aunque no toda cámara dentro de la aplicación representa un proyector.

Estas cámaras además pueden representar puntos de vista útiles para organizar el *show*, posicionar los objetos tridimensionales y desde donde simular lo que visualizaría un espectador.

Las cámaras permiten ajustar sus parámetros, mediante operaciones comunes a cámaras virtuales en otros paquetes de animación por software, y a cámaras cinematográficas.

Estas operaciones representan movimientos de cámara llamados *Roll*, *Orbit*, *Dolly* y *Pan*.

*Roll* se refiere al movimiento en donde la posición de la cámara y el punto de vista son fijos y esta rota a través del eje formado por ellos.

*Orbit* es cuando la cámara *orbita* alrededor de su punto de vista. Es decir, la posición de la cámara cambia pero siempre visualizando el mismo punto y manteniendo la distancia a este.

*Dolly* es el movimiento en donde se mantienen el punto y la dirección de vista pero se mueve la posición de la cámara acercándose y alejándose del objetivo.

Por último, *pan*, es el llamado *paneo* que es cuando se mueve la cámara cambiando la posición y el punto de vista pero manteniendo un paralelismo con la dirección.

Aparte de estos parámetros de ubicación, también existe el parámetro FOV (Field Of View) o campo de vista que es el ángulo de apertura de la cámara. Este parámetro debe coincidir con el FOV del proyector para ajustar correctamente las escenas con respecto a lo que muestra la cámara en pantalla.

Mediante estas operaciones es que las cámaras virtuales de VMT ajustan sus parámetros para que la proyección de objetos tridimensionales coincida con las superficies proyectadas.

Hay que destacar que las cámaras proveen un punto de vista solo para los elementos tridimensionales. Los *quads* no dependen del punto de vista de la cámara y se representan en un sector de la pantalla relativo a esta, por tanto, estos *quads* no alteran su posición en pantalla al ajustar los parámetros de la cámara.

Estos *quads* se organizan en *layers* o capas y en grupos.

Las capas sirven para organizar el dibujado de los *quads*. Cada capa pertenece a una única cámara de manera de representar las zonas a cubrir por un solo proyector. A su vez las capas permiten operar sobre ellas de manera de transformar todos los *quads* contenidos de forma coherente. Esto último es útil para ajustar la proyección de los *quads* en caso de mover el proyector levemente y que esta proyección deje de coincidir con las superficies.

Los *quads* se organizan también en grupos. Cada *quad* puede pertenecer a más de un grupo e incluso *quads* de distintas capas pueden pertenecer al mismo grupo.

Es necesario que un *quad* pertenezca a uno de estos grupos para poder aplicar efectos y texturas.

La idea detrás de esto es proveer un nivel más de abstracción para permitir mapear texturas a superficies cubiertas por más de un *quad* de la misma forma que se aplicarían a un solo *quad*.

Los grupos también permiten variar de forma limitada la forma en que se mapean las texturas a los *quads* componentes. Estas pueden ser mapeadas por cara en donde a cada uno de los *quads* se le mapeará toda la textura, o también de forma “plana”. Esto último significa que la textura será mapeada al conjunto entero de *quads* como si estos fueran trozos de un *quad* de mayor tamaño compuesto por todos ellos.

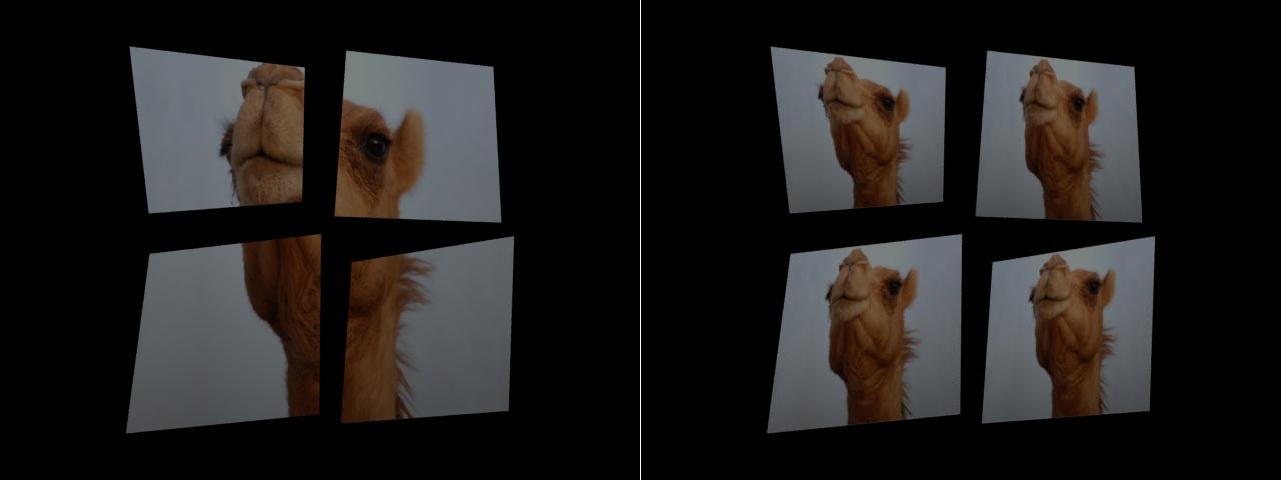


Fig.X: Izquierda: Proyección plana. Derecha: Proyección por cara.

Con estas variantes de proyección se le dan más posibilidades a un artista para lograr efectos interesantes.

Es también en el motor gráfico que tienen lugar los efectos posibles a ejecutar durante el *show* de *video mapping*.

El motor acepta como un efecto la asignación de texturas en forma de imagen o video ya sea a grupos de *quads*, o a objetos tridimensionales. Este es el efecto más básico pero también el más potente para realizar una gran variedad de resultados al mapear texturas sobre superficies.

La variedad y la complejidad está en la creación de los recursos de imagen o video que se deseen mapear. Al permitirse mapear estas texturas sobre objetos tridimensionales, no es necesario distorsionar la forma de los videos sino que será el motor que los adaptará a la forma del objeto que está siendo mapeado.

Otro efecto que soporta el motor es el llamado *fade* que realiza una animación en el color de un material, desde un color inicial hasta el resultado final.

Este efecto puede ser combinado en un objeto que ya disponga de una textura y así permitir por ejemplo su atenuación o resaltado.

La duración del *fade* es configurable para cada una de las instancias del efecto.

El otro efecto soportado es la animación de la posición de un objeto tridimensional. Esta animación es lineal entre un punto de origen y uno de destino y la duración de esta es configurable también.

Hay que aclarar que mientras los efectos no están siendo ejecutados, se mantiene el estado con el que ya venía la escena.

Un punto de extensión en el motor es la creación de más efectos genéricos como pueden ser animación de rotación, escala, modificadores de objetos tridimensionales, etc.

Como se mencionó anteriormente, una de las tareas principales del motor gráfico es la representación en pantalla o la proyección de los elementos gráficos.

La implementación del dibujado en si se realiza en un bucle llamado *main loop* o bucle principal, que se encarga de actualizar todos los objetos de la escena de ser necesario, procesar los eventos pendientes y finalmente dibujar el resultado. Esto es un proceso cíclico que se repite varias veces por segundo.

En cada uno de estos ciclos se generan los *frames* que son las imágenes que se muestran como resultado final.

Para dar la ilusión de continuidad, la frecuencia de estos ciclos debe ser alta. En el caso de *vmt\_engine* esta es de 60 ciclos por segundo.

Para que esta frecuencia se logre efectivamente es necesario que el procesamiento de cada uno de estos ciclos sea eficiente. De no ocurrir esto se podrían ver saltos o pausas en la reproducción del *show*.

Esta eficiencia se logra reduciendo el costo de procesamiento en las tareas más consumidoras del *main loop*.

Básicamente el *main loop* implementado funciona de la siguiente manera:

Pseudo-código del *main loop*

Posicionar cámara

***para todo*** objeto3d en la escena:

***para toda*** selección de caras:

cargar material

dibujar caras con el material

fin

fin

Reset cámara

para todo quad

cargar material

dibujar quad

fin

Los cuellos de botella en este tipo de *main loop* son varios.

La carga de materiales implica copiar una textura a la zona de la memoria de video correspondiente para utilizarla. Esto podría ser un proceso lento si no se disponen las texturas precargadas en memoria. Es por esta razón es que las texturas se cargan desde un comienzo en memoria y luego se utilizan referencias para identificarlas y asociarlas a una unidad de textura de OpenGL [www.opengl.org/wiki/Texture] para finalmente poder utilizarla.

Otra optimización es el uso de *shaders* para el cálculo del resultado visual final de las texturas combinadas con el color ambiente y la iluminación de la escena.

Los *shaders* son programas que se ejecutan en los procesadores gráficos y tienen como ventaja que son altamente paralelizables, permitiendo realizar cálculos de forma más rápida.

VMT utiliza el lenguaje de *shaders* GLSL [www.lighthouse3d.com/opengl/glsl].

La visualización fluida de un *show* de *video mapping* es difícil de prever ya que, como se puede ver en el main loop, los tiempos de ejecución dependen de la cantidad de materiales, caras, objetos y *quads* incluidos en la escena.

Si se utilizan videos en las texturas, la compresión y resolución de estos también afecta la fluidez del *show*. Para lograr mayor fluidez estos videos deben estar descomprimidos o con una baja tasa de compresión.

Lo mismo ocurre para formatos de imagen con compresión aunque esto afecta en menor medida a la reproducción del *show*.

### Interfaz de usuario

GUI

### Multi proyector

Distribución Nodes y Server

Modo dual, triple en la misma tarjeta

Utilizando varias instancias de clientes

### 

### Tratamiento de malla

Algoritmos para tratamiento de mallas

La mayoría de los dispositivos de captura de información 3D que existen actualmente entregan la información en forma de nube de puntos (ref a Kyle, kinect y Sensores del chino). Es por ello que previo a la manipulación de la información tridimensional, es necesario procesar dicha nube de puntos para convertirla a formatos más manejables, como por ejemplo mallas triangulares.

Un típico procesamiento de malla, ya muy estudiado e implementado en bibliotecas como VcgLib o CGAL [7] dada una nube de puntos de entrada, realizar un sub-muestreo y suavizado de la misma, calcular las normales en cada punto de la nube, y finalmente aplicar algoritmos de reconstrucción de malla.

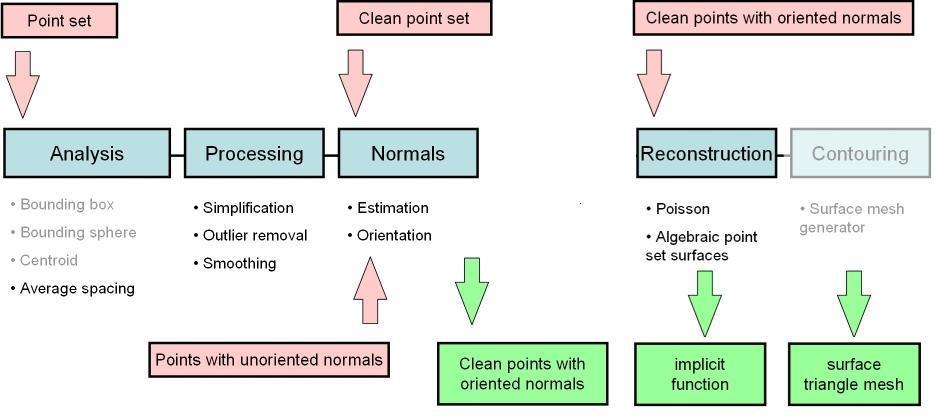


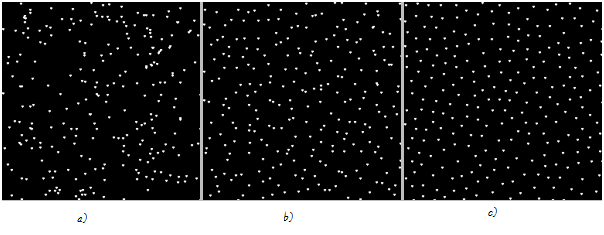
Figura: Típico flujo para el procesamiento de nubes de puntos (fuente: CGAL[7])

Para la implementación de este modulo, se utilizaron algoritmos incluidos en VcgLib. Nos apoyamos fuertemente en MeshLab [8] para visualizar y evaluar los resultados esperados. Particularmente se utilizaron los algoritmos de muestreo *Poisson-disk* para reducir y normalizar los puntos de la malla inicial, *Normal Extrapolation* para el cálculo de normales y reconstrucción de superficies de *Poisson* para la reconstrucción de la malla final.

Muestreo Poisson-disk

El muestreo o sorteo de variables aleatorias es una técnica utilizada para una gran variedad de aplicaciones graficas, incluyendo dibujado (*rendering*), procesamiento de imágenes y de geometrías.

Particularmente, el muestreo Poisson-disk se utiliza para ubicación aleatoria de objetos en mundos artificiales, algoritmos de texturas procedurales y procesamiento de geometrías o mallas. Esta técnica genera conjunto de puntos con las siguientes propiedades*: puntos suficientemente juntos, pero con la restricción de no estar más próximos unos de otros que una distancia mínima R predeterminada*.



*Figura a) Posición x e y generadas aleatoriamente. Figura b) Imagen dividida en celdas. Puntos aleatorios generados en cada celda. Figura c) Muestreo Poisson-disk en 2 dimensiones.*

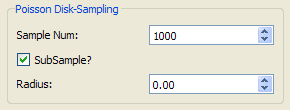
La idea básica de este algoritmo es generar puntos alrededor de los ya existentes en la muestra, y validar si pueden ser agregados al conjunto final en caso de no violar la regla de la mínima distancia a los vecinos. Se genera una grilla en 2 o 3 dimensiones dependiendo del escenario de aplicación, en la cual cada celda contendrá al final del proceso a lo sumo un punto. Una grilla adicional es utilizada para realizar búsquedas rápidas, y dos conjuntos de puntos son mantenidos durante el procesamiento para poder diferenciar los que han sido generados y los que aún necesitan procesamiento.

La implementación de VcgLib utilizada recibe 3 parámetros:

-La cantidad de puntos en la muestra. En este caso el radio o parámetro de cercanía es calculado en base a este parámetro.

-El radio, que es a su vez utilizado para calcular el tamaño de la muestra optimo en base a la malla inicial.

-Sub muestreo: indica si la muestra de Poisson es un subconjunto de la muestra inicial o si se deberán generar nuevos puntos aleatoriamente.

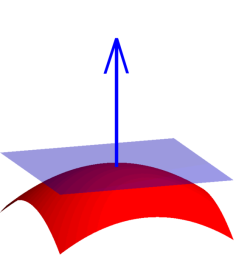


Reconstrucción de normales

Este algoritmo computa las normales en cada elemento de un conjunto de puntos sin la necesidad de explorar la conectividad de los triángulos. Por ello es que es muy útil para objetos tridimensionales sin información de caras (*faces*).

Se detalla un pseudocódigo del método:

***Paso 1: Identificar los planos tangentes para aproximar localmente la superficie y estimar así los vectores normales***

 Para cada vértice:

Calcular el centro geométrico (*centroide*) del plano tangente en el punto como el promedio de los K puntos más cercanos.

Calcular la normal asociada al centro geométrico. Se utiliza la matriz de covarianza en el punto contemplando los mismos K vecinos más cercanos de la muestra y los valores y vectores propios de la matriz de covarianza. Finalmente, ordenando los vectores propios, la estimación del vector perpendicular corresponde al vector propio de menor valor. Este método es conocido como PCA (*Principal Component Analisys*).

***Paso 2: construir grafo donde cada punto está conectado a los K vecinos más cercanos (grafo de Riemannian)***

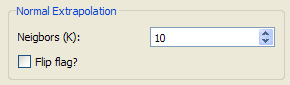
Se crea un grafo en cuyos nodos se guardan todas las aristas a los K vecinos más cercanos. Cada arista se pesa con el valor absoluto del producto escalar de la normal en el punto con la normal en cada uno de los K vecinos:

*fabs(nodoActual->normal . K\_Vecinos[n]->normal)*

***Paso 3: calcular el árbol de expansión mínimo (MST) sobre el grafo de Riemannian y recorrerlo para orientar las normales***

Dado un grafo conexo, no dirigido, con sus aristas con un peso asignado, se llama árbol de expansión mínimo al sub-grafo con forma de árbol que conecta todos los nodos con un peso total mínimo. Contiene todos los nodos del grafo inicial. El grafo de entrada es el construido en el paso enterior y se utiliza el algoritmo de Kruskal, uno de los varios algoritmos glotones que resuelven el problema de encontrar un árbol de expansión mínima de un grafo (referencias).

Una vez construido el MST, lo único que se hace es recorrer el árbol en orden y corregir el sentido de los vectores normales (multiplicando por *-1.0f*) en caso de ser necesario. La condición para efectuar dicha corrección se basa en el ángulo del nodo siendo inspeccionado versus todas las direcciones de las normales de los vecinos conectados a dicho nodo.



Reconstrucción de malla de Poisson

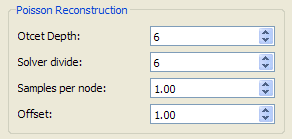
Finalmente, para reconstruir la malla a partir de la nube de puntos y sus normales se utiliza el algoritmo de Reconstrucción de Poisson.

Se computa una función indicador en 3 dimensiones definida de la siguiente forma:

Luego, se obtiene una reconstrucción de la superficie mediante la extracción de la *ISO-surface* (superficie de nivel en 3 dimensiones) al nivel apropiado.

La clave está en que hay una estrecha relación entre los puntos orientados (con sus normales) de la muestra y la función indicador de la muestra. Específicamente el gradiente de la función indicador es un espacio de vectores que valen cero casi en todo el espacio excepto en puntos cercanos a la superficie, donde es igual al vector normal a la superficie.

Es por eso que puntos orientados, pueden ser vistos como muestras del gradiente de la función indicador del modelo tridimensional en cuestión. Y es por este mismo motivo que el problema de reconstrucción de una malla puede ser visto como un problema de Poisson estándar: computar la función escalar F cuyo Laplaciano (o divergencia del gradiente) se iguala a la divergencia del espacio de vectores de las normales.

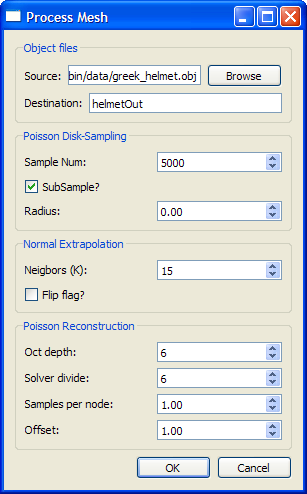


Pruebas y resultados

Para validar el correcto funcionamiento de esta técnica utilizando los tres algoritmos descritos, utilizamos una malla inicial de 5021 vertices y 9608 caras triangulares. Cabe destacar que si bien se ha mencionado que la malla de entrada debe ser simplemente una nube de puntos, se pueden utilizar mallas con caras, solo que estas serán ignoradas e incluso removidas de la malla de salida del primer paso del procesamiento (Poisson-disk sampling).

Parámetros iniciales

Luego de experimntar con varios juegos de datos iniciales durante varias ejecuciones del procesamiento, se fijaron de manera personalizada para la malla de entrada algunos parámetros clave. Dado que la muestra inicial tiene alrededor de cinco mil puntos, elegimos cinco mil como cantidad de muestras para el algorimso de Poisson-disk. Luego, para la extrapolación de normales se utilizaran K=15 vecinos para la toma de decisiones locales de aproximación.



Resultados

La aplicación de estos algoritmos resultó ser lo esperado, al menos en cuanto a la parte estructural de cada malla que va siendo procesada en cada paso. No se llego a procesr mallas de ambientes tridimensionales escaneados para luego ser mapeados con la herramienta. **Creemos que esto debería ser validado en una etapa posterior de este proyecto ya que ese es el principal contenido del presente modulo.**

|  |  |  |
| --- | --- | --- |
| **Fase** | **Vértices** | **Caras** |
| Nube inicial | 5021 | 9608 |
| Luego Poisson-disk | 1776 | 0 |
| Luego Extrapolación Normales | 1776 | 0 |
| Luego de Poisson-recontruction | 1959 | 3910 |

Tabla: Comparacion de estructura de malla de entrada y salida de cada fase.

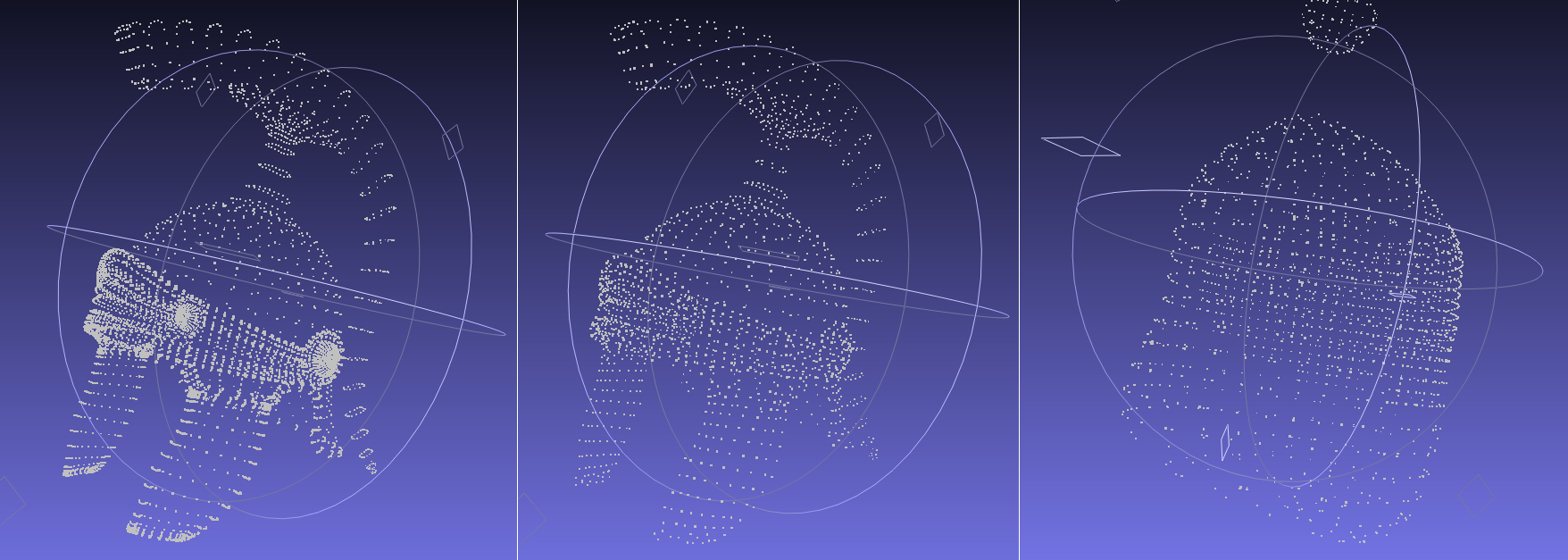


Figura: 1) Nube de puntos inicial con 5021 vértices. 2) Resultado de muestreo Poisson-disk con 1776 vértices. 3) Luego de extrapolar normales y reconstruir la malla con 1959 vértices y 3910 caras.

Se observaron buenos tiempos computacionales de respuesta. Si bien la malla utilizada no es de un tamaño considerable, estamos hablando de algoritmos de orden relativamente alto.

### Calibración

Calibración 2D (homografía)

Calibración 3D (ajuste de cámara)

### Efectos

Efectos estáticos (definición)

Orientado a tiempo/eventos de teclado

# Conclusiones

## Dificultades encontradas, limitaciones y posibles mejoras

### Adquisición

### Calibración

### Conclusiones de la autoevaluación

## Prueba de desempeño

## Conclusiones

# ANEXO I – Eventos realizados

# ANEXO II - Casos de uso relevantes

[ Agregar/editar cámara, agregar efecto, agregar quad, test efecto, calibrar, schedule effect, play show, agregar audio, save/load ]

# Glosario

# Bibliografía

[1] Computer and Robot Vision VOLUME II

Robert M.Haralick Linda G.Shapiro

Capítulo 16

[2] http://web.yonsei.ac.kr/hgjung/Ho%20Gi%20Jung%20Homepage/Lectures/2009%20Fall%20Computer%20Vision/Handouts/7%20Structured%20Lighting.pdf

[3]IEEE TRANSACTION ON PATTERN ANALYSIS AND MACHINE INTELIGENCE, VOL. PAMI-5, NO.2 MARCH 1983

A Perspective on Range Finding Techniques for Computer Vision

[4] IEE Transactions On Systems, Man, and Cybernetics, vol19, no.6,November/dicember 1989 Structure from Stereo- A Review

<ftp://ftp.iutbayonne.univ-pau.fr/pub/perso/Gim/luthon/deposite/stereo25.pdf>

[5] RECENT PROGRESS IN CODED STRUCTURED LIGHT AS

A TECHNIQUE TO SOLVE THE CORRESPONDENCE

PROBLEM: A SURVEY

<http://www.google.com.uy/url?sa=t&source=web&cd=1&ved=0CBcQFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.9.5774%26rep%3Drep1%26type%3Dpdf&rct=j&q=10.1.1.9.5774&ei=Q20ZTPySNcGBlAeRpcmxCw&usg=AFQjCNFAJSc7jcpjmM0bCoMVyeNAiqIkkQ>

Pattern codification strategies in structured light systems

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.6920&rep=rep1&type=pdf>

Joaquim Salvi Jordi Pagès Joan Batlle

[6] http://ftp.isaatc.ull.es/pub/misc/ja2002/Comunicaciones/ja02\_058.pdf

[7] VcgLib "*Visual computer graphics Library*”. <URL:http://vcg.sourceforge.net/tiki-index.php>. Biblioteca portable escrita en C++ para manipulación, procesamiento y despliegue con OpenGL de mallas triangulares. Liberada bajo licencia GPL por parte del “*Visual Computing Lab* “ (VCGLab) del "Institute of the Italian National Research Council " <URL:http://www.isti.cnr.it>.

[8] MeshLab

Software de código abierto para manipulación de mallas tridimensionales en varios formatos.

<http://meshlab.sourceforge.net/>

# Índice de figuras

# Índice de cuadros