Algoritmos para tratamiento de mallas

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

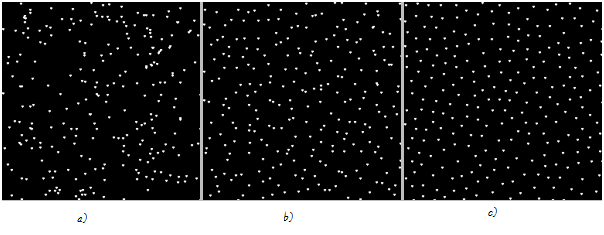
Para el propósito de nuestro proyecto, dada una nube de puntos alcanza con realizar un sub-muestreo de la muestra inicial, calcular o recalcular la normales en cada punto y finalmente aplicar algoritmos de reconstrucción de malla.

Para la implementación de este modulo, se utilizaron algoritmos incluidos en VCGLib, biblioteca portable escrita en C++ para manipulación, procesamiento y despliegue con OpenGL de mallas triangulares. 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 (Poisson-disk Sampling)

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. Lo más interesante de esta técnica es que genera conjunto de puntos con buenas propiedades*: puntos suficientemente juntos, pero con la restricción de no estar más próximos unos de otros que una distancia mínima R predeterminada*.

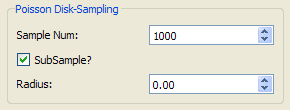


*Figura a) Posición x e y generadas aleatoriamente. Figura b) Imagen dividida en celdas. Puntos aleatorios generados en cada celda. Figura c) Muestreo Poisson-disk en 2 dimensiones.*

La idea básica es generar puntos alrededor de los ya existentes en la muestra, y validar si pueden ser agregados a la muestra 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 de la 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 de los puntos, y dos conjuntos de puntos son mantenidos durante el procesamiento, para poder diferenciar los que han sido generados y los que aun 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 de cada vértice 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).

La idea básica del algoritmo es computar

Paso 1: identificar los planos tangentes para aproximar localmente la superficie

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 (creada con el método de Jacobi), 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*). Se está haciendo uso de la perpendicularidad de los vectores propios entre sí.

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

// Step 3: compute the minimum spanning tree (MST) over the Riemannian graph (we use the Kruskal algorithm)

// compute for each plane the list of sorting edges

// Traverse the incident\_edges vector and build the MST

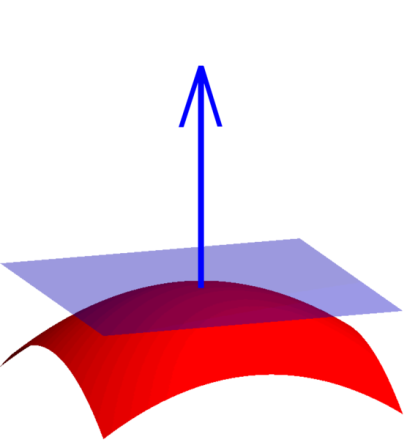
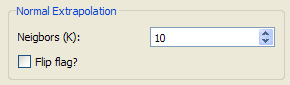
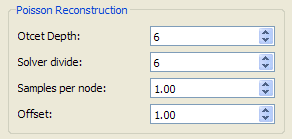


Figure - Vector normal



* Reconstrucción de malla



* Resultados

