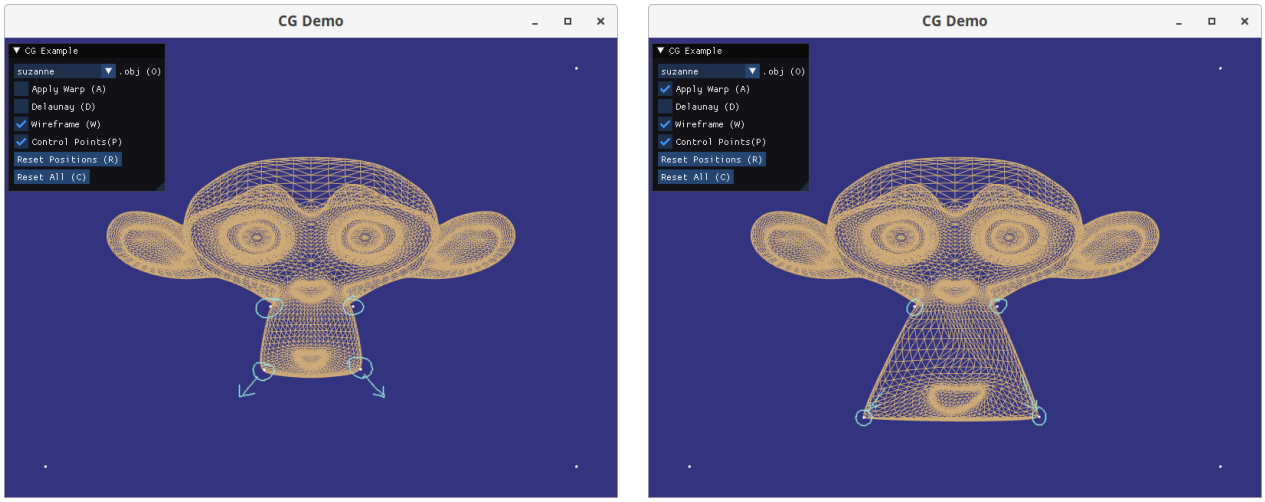
Computación Gráfica - TP: Warping

1. Resumen de tareas

1. Completar la implementación de la función `calcularPesos` (en `utils.cpp`)
2. Completar la implementación de la función `warpPoint` (en `main.cpp`)
3. Entender y documentar el método `Delaunay::enQueTriangulo` (en `Delaunay.cpp`)

2. Consigna detallada

El objetivo de este práctico es lograr deformar todo un modelo a partir de marcar y ”tirar” de unos pocos puntos. El usuario elige puntos sobre (o cerca de) la geometría original, y al arrastrarlos, la geometría debe deformarse como si se tirara de esos puntos.



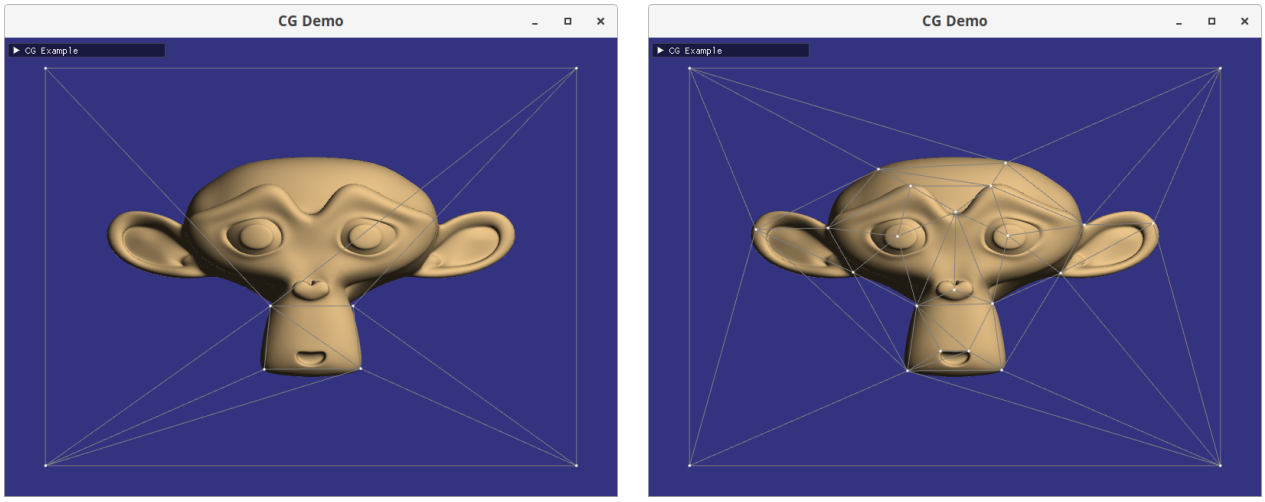
La geometría del modelo está definida por un conjunto de primitivas (triángulos). El principal objetivo del práctico es implementar la función que ”mueve” o ”transforma” los vértices de las primitivas para lograr el efecto deseado. En el código, **la función `warpPoint` es la que dado un punto lo transforma, y deberá ser implementada por el estudiante**. La función `applyWarp` toma cada uno de los puntos originales de la geometría e invoca a `warpPoint` para generear una nueva geometría deformada (esta función ya está implementada).

Para simplificar el práctico, la deformación es 2D (para cada punto solo se deforman las coordenadas *x* e *y*, *z* debe quedar como estaba). El usuario define entonces puntos en 2D clickeando sobre la ventana (en el código, por conveniencia los puntos 2D están representados por instancias de `glm::vec3` con componentes *z=0*). Los puntos se mueven simplemente arrastrando con el ratón. El programa arma dos triangulaciones con los puntos que marca el usuario: una con las posiciones originales de los mismos (es decir, sobre la malla sin deformar) y otra con las posiciones modificadas. El alumno deberá aprovechar estas triangulaciones para calcular las nuevas posiciones de los vértices de las primitivas.

Ayuda 1: La función `warpPoint` recibe 2 triangulaciones. Esto es, 2 vectores de puntos y 2 de triángulos. Para resolver el problema solo deberá usar los 2 vectores de puntos (el del argumento `delaunay0` contiene los puntos

originales, el de `delaunay1` contiene los punto desplazados) pero solo una una de las 2 listas de triángulos (¿cuál? ¿por qué?)

Ayuda 2: si no se le ocurre un método para lograr la deformación, piense en la siguiente pregunta: ¿por qué a los pesos de la interpolación afín se los suele llamar "coordenadas" baricéntricas?



Las triangulaciones se modelan en el código como instancias de una clase `Delaunay`. Este es el nombre de un tipo de triangulación con ciertas propiedades que resultan convenientes para hacer interpolaciones. No hace falta que el alumno conozca ni entienda en este momento ninguna de las particularidades de estas triangulaciones (se estudiarán en otra unidad). Podría considerarse una triangulación cualquiera. No hace falta que el alumno vea ni analice el código de dicha clase, con la sola excepción de un único método que se mencionará más adelante.

La clase `Delaunay` guarda un vector de puntos/vértices de la triangulación (serán los que clickee el usuario), y un vector de triángulos. Cada triángulo es un conjunto de tres índice (es decir que indica 3 puntos del vector de puntos que deben conectarse). Cada triángulo además guarda los índices de los 3 otros triángulos que tiene por vecinos (o `-1` si no hay vecino). Estarán ordenados de forma tal que siempre en un triángulo el vecino *k-ésimo* es el opuesto al vértice *k*.

Los métodos `Delaunay::getPuntos` y `Delaunay::getTriangulos` permiten consultar estos vectores. **La clase contiene además un método** `Delaunay::enQueTriangulo` que dado un punto en el espacio retorna el índice del triángulo que lo contendría. El alumno puede utilizar este método para implementar la transformación de los vértices; pero además **deberá analizar su funcionamiento y agregar al mismo los comentarios que crea necesarios**.

Por último (aunque esto será lo primero a resolver), al iniciar el práctico la triangulación no se construirá correctamente. **El alumno deberá completar la implementación de la función** `calcularPesos` para que `Delaunay` funcione correctamente. Esta función debe, dados tres vértices de un triángulo (`x0`, `x1` y `x2`), y un 4to punto (`x`) calcular los pesos para obtener ese 4to punto como combinación afín de los 3 primeros. Debe utilizar un cálculo que prevea el caso de extrapolación (es decir, que algún o algunos pesos den negativos cuando el 4to punto está fuera del triángulo).