

Computación Gráfica - TP: Pez

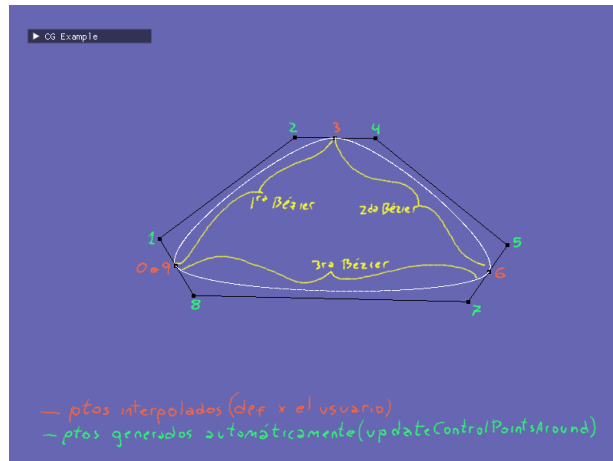
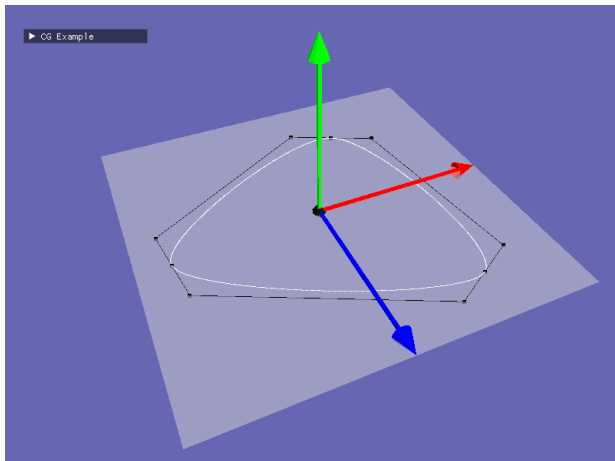
1. Resumen de tareas

1. Definir automáticamente los puntos de control no interpolantes de la spline, de modo que la curva resulte *suave*.
2. Generar la transformación adecuada para que el pez se desplace siguiendo la trayectoria de la spline.
3. Analizar la variabilidad de la velocidad del pez a lo largo de la trayectoria y en caso de presentar problemas proponer (no implementar) posibles soluciones.
4. *Opcional*: Analizar cómo se logra animar la deformación del pez.

2. Consigna detallada

2.1. Definición de la trayectoria

El objetivo del práctico es poder definir automáticamente una curva a partir de un conjunto de puntos a interpolar. El código implementa una clase `Spline` para representar una sucesión de curvas de *Bezier* de grado 3. Debe pensar que la clase `Spline` guarda un conjunto de puntos de control, que tomados de a 4 con 1 de solapamiento y de forma cíclica, definen los tramos como curvas de *Bezier*. Por ejemplo, si la *Spline* contiene 9 puntos, los puntos 0 a 3 definen el primer tramo, los puntos de 3 a 6 el segundo (notar que el 3 es el último del primer tramo, y el primero del segundo), y los puntos de 6 a 0 el tercero (no habría punto 9, sino que después del 8 se conecta otra vez con el 0).



Los puntos 0, 3 y 6 son los interpolados (los que define libremente el usuario), mientras que el resto (2 puntos de control interiores en cada tramo de *Bezier*) se deben calcular automáticamente de forma que la curva resulte *suave* (pregúntese antes de plantear un método: ¿qué significa *suave*?).

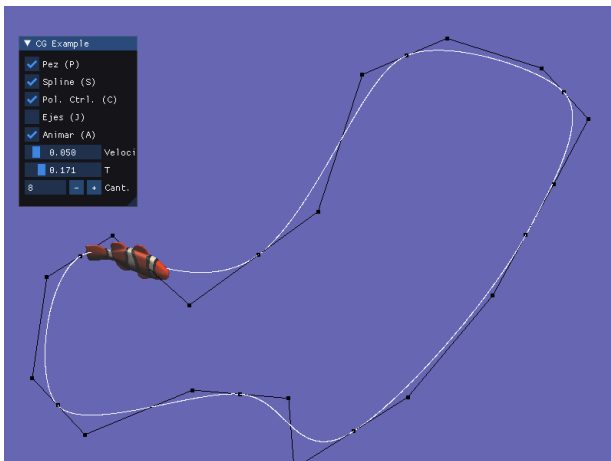
Para acceder al punto de control i -ésimo, la clase `Spline` ofrece los métodos `glm::vec3 Spline::getControlPoint(int i) const` (para consultar su posición), y `void Spline::setControlPoint(int i, glm::vec3 p)` (para modificarlo). En ambos puntos puede utilizar índices fuera del rango válido y el método se encargará de ajustarlos cíclicamente. Por ejemplo, si en una curva con 9 puntos (donde los índices válidos serían en principio de 0 a 8) si ingresa el índice 9 se obtienen el primer punto (0), o si ingresa el índice -1 se obtiene el último (8).

El programa cliente define en *main.cpp* una función `void updateControlPointsAround(Spline &spline, int ctrl_pt)` que debería recalculan las posiciones de los dos puntos de control no interpolados alrededor del punto i -ésimo (el previo, $i-1$, y el siguiente, $i+1$), a partir de la posición de los puntos interpolados (por ej, a partir de los puntos $i-3$, i , $i+3$). Esta función por defecto no hace nada, y es tarea del alumno completar su código. Notar que la función recibe a la *Spline* por referencia, así que debe modificar los puntos de control a través de ese objeto y no retornar nada.

Por cómo está definida la curva y la manipulación que permite la interfaz, los puntos de la *Spline* siempre estarán en el plano $y=0$.

2.2. Movimiento del pez

Una vez definida correctamente la *Spline*, el programa debe utilizarla para posicionar el modelo del pez de forma que avance recorriendo la misma. Para ello, el programa cliente define una variable t que avanza con el tiempo recorriendo posibles valores para el parámetro de la *Spline* (cuando llega al valor máximo vuelve a 0). En cada frame, se debe utilizar el valor de t para generar la transformación adecuada. Para esto, se define en *main.cpp* una función `glm::mat4 getTransform(const Spline &spline, double t)` que recibe la *Spline* y el t actual y debe retornar la matriz adecuada.



Para evaluar la *Spline*, puede utilizar los métodos `glm::vec3 Spline::at(double t) const` y `glm::vec3 Spline::at(double t, glm::vec3 &deriv) const`. Ambos métodos reciben como argumento el valor del parámetro t (que va de 0 a 1¹) y retornan el punto de la curva correspondiente a dicho parámetro. El segundo método además guarda en el argumento *deriv* (que pasa por referencia) el vector derivada de la curva en ese punto.

Para armar la matriz, se recomienda pensar en las columnas como vectores de la nueva base (las 3 primeras) y nueva posición del origen (la cuarta). El código inicial muestra cómo armar una matriz a partir de estos 4 vectores/puntos. Notar que el modelo del pez está centrado en el origen del sistema de coordenadas, alineado con el eje x , y mide 2 unidades. Puede activar la visualización de los ejes para referencia.

¹La clase *Spline* internamente considera que t varía 1 unidad en cada tramo. Si la spline tiene 5 tramos de Bezier, entonces t debería ir de 0 a 5. Pero para que el cliente de la clase no deba ajustar su parámetro a un rango variable, la clase espera un t normalizado entre 0 y 1, e internamente lo mapea. Para ello multiplica el t normalizado por la cantidad de tramos para así obtener el número de tramo a utilizar (la parte entera del resultado) y el t local para dicho tramo (la parte decimal).

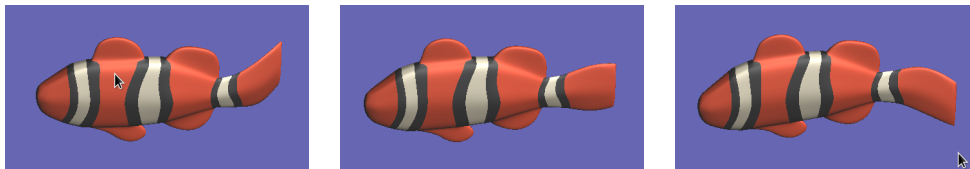
2.3. Forma de la curva y velocidad de Avance

Cuando tenga resuelto el movimiento del Pez, analice su solución preguntándose:

1. ¿Es *adecuada* la forma de la curva que se genera?
 - De acuerdo al método elegido/desarrollada para calcular los nuevos puntos, la curva podría ser *demasiado* plana, *demasiado* abierta, o presentar características como *overshooting*.
 - Cuánto es *demasiado*, o si el *overshooting* es un *bug* o una *feature* es materia de discusión (discútalos con sus compañeros de grupo para defender luego su propuesta ante el docente).
 - Cualquiera sea el método, en los casos fáciles u obvios, el resultado debería ser previsible y similar. Por ejemplo, si coloca los puntos a interpolar equidistantes sobre un círculo todos los métodos deberían aproximar el círculo (si el *círculo* se parece más a un *polígono* probablemente su método no sea correcto).
2. ¿Hay variaciones de velocidad? (puede ser bruscas o suaves).
 - Si las hay, ¿a qué se deben?, ¿sería correcto corregirlas? ¿cómo? (solo debe pensar un algoritmo de solución, pero no es necesario implementarlo ya que podría requerir demasiadas modificaciones).
 - Para generar casos donde se resalten estas diferencias, genere curvas donde los puntos a interpolar tengan distancias muy diferentes entre ellos (que algunos estén muy cerca entre sí y otros muy lejos).

2.4. Deformación de la geometría del Pez

Cuando se habilita la animación, la geometría del pez se deforma para simular el movimiento que hace este con su cola para avanzar. La animación se genera modificando las posiciones de los vértices en el vertex shader (archivo *fish.vert*) en cada frame. Analice esta implementación.



¿Es conveniente implementar la animación de esta forma? ¿O es mejor tener un archivo de entrada que defina genéricamente la animación de la malla?

Nota: Definir una animación genéricamente implica usualmente que en el/los archivos de donde se carga el modelo, se defina para cada vértice más de una posición. Se definen las posiciones para ciertos fotogramas denominados *clave* y el sistema deberá interpolar las posiciones intermedias para los demás fotogramas. En este ejemplo podríamos tener los vértices para representar el pez arqueado hacia un lado y hacia el otro y generar la transición interpolando.