# Parcial - Ejercicio 1 (de 3)

Se tiene un malla como la de la imagen de abajo, para un personaje de un videojuego. Como el personaje debe ser re-heavy re-jodido, tendrá un tatuaje (un dibujo de La Púa del Destino) en el brazo izquierdo, cerca del hombro. Indique cómo aplicaría el tatuaje como textura sobre el modelo, qué tipo de mapeo utilizaría (incluya todas las ecuaciones necesarias para implementar el mismo), qué métodos de mezcla y/u otras configuraciones de la aplicación de textura deberá considerar, así como también si fuera necesaria alguna particularidad de la imagen.


Nota: la malla de la versión final podría estar más refinada, así que debe descartar un mapeo completamente manual, sería demasiado trabajo.

Fuentes de las imágenes:
- Rocker 3d low poly by ~toman90 on deviantART
- Pick Of Destiny Stickers | Redbubble ((c) asecrist)

# Parcial - Ejercicio 2 (de 3)

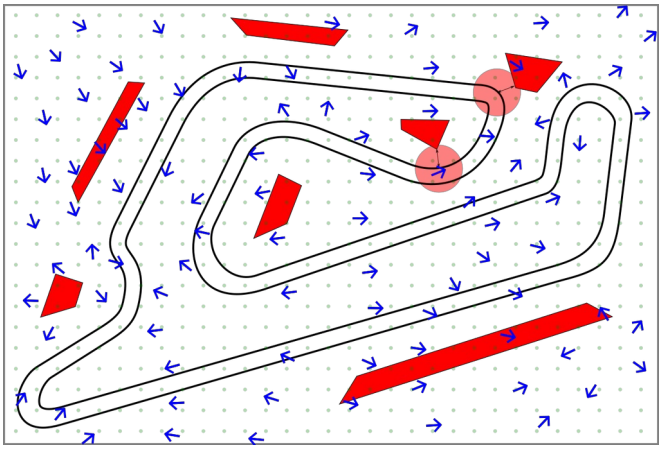
Usted ha sido contratado para diseñar un software que ayudará a analizar posibles circuitos para el nuevo campeonato mundial de Fórmula Inc (autos impulsados por energía Increíble). Lo implementarán un conjunto de programadores a su cargo muy buenos para aprender APIs, pero con cero conocimiento de rendering y de geometría, por lo que deberá especificarles todos los detalles necesarios para implementar cada algoritmo (pasos, ecuaciones, etc... todo menos la API, ya que van a buscar las funciones de OpenGL por su cuenta).

1. Para analizar el drenaje natural del terreno se necesita que el programa dibuje, sobre una foto satelital del mismo, flechas indicando hacia dónde drena el agua en caso de lluvia de acuerdo a su inclinación. Como datos, se tienen (además de la foto) la altura exacta del terreno para cada punto en una grilla regular de NxM puntos sobre el bounding box del terreno.

2. Por seguridad, ninguna tribuna debe estar a menos de 30 metros de la línea de carrera. El programa debería verificar si esto se cumple, y si no es así indicar dónde está el problema. Como datos, tiene 2 splines de beziers de 3er grado cerradas que definen los bordes derecho e izquierdo del circuito, y por cada tribuna las coordenadas de los 4 puntos de sus vértices (vistas de arriba, todas tienen forma cuadrilateral). Las coordenadas de ambas cosas están en metros.

Nota: Recuerde que a estos programadores hay que explicarles casi todo. Por ej, no les puede decir "dibujen una flecha” porque no hay ninguna función glDrawFlecha en OpenGL, solo les puede pedir que dibujen primitivas (puntos, segmentos, triángulos, cuadriláteros, eso sí lo van a saber dibujar porque lo provee la API).

Ejemplo de los datos de entrada y a calcular:



Los puntos verdes serían los puntos donde se conoce la altura, y las flechas azules (las que hay que dibujar en 1) indican para dónde "baja” el terreno. Los cuadriláteros rojos serían las tribunas, y los círculos tienen radio 50m indicando dos lugares donde están demasiado cerca de la pista (lo que hay que detectar en 2; aunque puede marcarse solo el segmento, no hace falta dibujar el círculo). Los límites de la pista están definidos por las dos curvas cerradas negras.

# Parcial - Ejercicio 3 (de 3)

La "complejidad de z" en una imagen mide cuantas veces se repintó cada mismo píxel al renderizar la misma. Por el algoritmo de z-buffer, si se renderizan los triángulos de atrás para adelante, cada nuevo triángulo tapa al anterior y entonces repinta donde ya había pintado el anterior. Si se renderizan de adelante hacia atrás, ocurre todo lo contrario, en las zonas donde se superponen dos triángulos, si se pinta primero el de adelante, los fragmentos de los demás se descartan, y entonces nunca se vuelve a repintar el mismo píxel.

Si bien la principal ventaja de usar este algoritmo (z-buffer) es que no hace falta ordenar, el resultado siempre es correcto; el tiempo de renderizado sí cambia ya que en el primer caso se sombrean muchos más fragmentos que en el segundo.

Se tiene una función que renderiza una escena y se quiere visualizar la complejidad de la siguiente forma: partiendo de la imagen de la escena convertirla a blanco y negro, y luego teñir cada pixel con un color que indique cuanto se repintó durante el renderizado. Se usará una escala que va desde azul (fragmentos que se pintaron solo una vez) a rojo (máxima cantidad de repintados), pasando por celeste, verde, y amarillo y naranja para valores intermedios.

Escala de color: 

Explique cómo podría generar esta imagen. Dispone de la rutina "drawScene()" que envía al pipeline todos los triángulos de la escena (con sus materiales y texturas, y sin hacer ningún otro truco especial). La sintaxis específica de OpenGL no es importante (invente un pseudocódigo sino recuerda), pero debe plantear el algoritmo para lograr esto, incluyendo detalles como las configuraciones del pipeline en cada pasada, la geometría a renderizar si usa algo además del drawScene, cómo transforma los valores de un buffer si es necesario (por ej, para pasar a blanco y negro, o para obtener el color a partir del número de repintados), etc. Ayuda: recuerde que puede pedirle a OpenGL pasar datos entre un buffer del framebuffer y un arreglo en ram, en ambos sentidos.