Lineamientos de las Soluciones al Parcial 1 de 2024

Ej1: rasterización vs. raytracing

- **Rasterización**
  - El paso importante es: por cada primitiva, proyectarla sobre el "plano de la imagen" para ver qué pixeles cubriría (justamente esto es "rasterizarla"), y luego para pixel que cubre sombrear y decidir si queda (o si otro lo tapa).
  - Se usa con modelos de iluminación "local" como Phong, y así resulta muy "barato" y eficiente para la gpu, por lo que es el habitual para tiempo real.
- **Raytracing**
  - El paso importante es: "por cada pixel tiro un rayo que parte del ojo y pasa por el mismo, y veo a qué primitiva choca primero"
  - Luego, para "sombrear", al tener la "maquina" de tirar rayos, puedo usar eso para calcular iluminación, de ahí que tenga mucho mejores resultados (será "global", sombras, reflejos, etc) y que sea mucho más caro (la cantidad de rayos, crece exponencialmente con el nro de rebotes a considerar). Por esto es más habitual en renderizado off-line.

Ej2: punto vs vert. vs frag. vs pixel

- **punto**: posición (coordenadas) en un espacio (por ej, *x,y,z* en 3D), nada más
- **vértice**: en el pipeline es un punto que define una primitiva (un segmento o un triángulo), y que además lleva información asociada (no es solo posición, lleva normal, color/material, coordenadas de textura, etc)
- **fragmento**: cada fragmento se corresponde con un pixel, pero todavía no lo es (puede que se descarte antes de llegar a la imagen final), y tiene información asociada (salió de una primitiva, hereda la información adicional de sus vértices).
- **pixel**: un pedacito/cuadradito de una imagen raster (un posición en la matrix, donde guardo un color).

Ej3: Rayo reflejado

- **Calcular x**: $v_l = l - p \quad \Rightarrow \quad x = p + (v_l \cdot n)n$ (si n está normalizado, sino dividir por $|n|^2$)
- **Pesos**: Si $r = \alpha_p p + \alpha_x x + \alpha_l l \quad \Rightarrow \quad \alpha_p = 0,\ \alpha_x = 2,\ \alpha_l = -1$
  - Se obtienen fácilmente si se dibujan las isolineas de cada α.
- **Dirección del rayo reflejado**: $d = \frac{r-p}{|r-p|}$

Ej4: Rasterización de un triángulo

Opción 1

- Por cada pixel de la imagen, calculo si (su centro) cae dentro del triángulo. Para eso calculo "la mitad" de las coordenadas baricéntricas (solo el numerador, el denominador no importa porque solo me interesa el signo). Incluir la fórmula (producto vectorial bien ordenado).
- Para interpolar los valores necesito los pesos de la interp afin, así que "completo" el cálculo de las coordenadas baricéntricas en los pixeles que estén dentro (ver abajo notas sobre la interpolación).
- Este método es muy simple y paralelizable... pero va a testear muchisimo pixeles que no van a estar en el triángulo (caso extremo, un triángulo grande y aplastado orientado con la diagonal de la ventana).
    - Se puede mejorar recorriendo solo dentro del bounding-box, y/o haciendolo jerárquico/recursivo empezar con "pixeles" más grandes para descartar muchos de una, pero no hacía falta aclarar esto.

Opción 2

- Usar la idea de scanlines: primero "rasterizar" el borde, y luego aplicar la idea del test del rayo en cada fila:
    - "rasterizar" el borde es calcular los fragmentos, pero no pintarlos, guardarlos en alguna lista o vector auxiliar. Hay que explicar brevemente el algoritmo (DDA de lineas).
    - el test del rayo se simplifica porque es horizontal y para un triángulo hay como mucho 2 "cortes", que van a ser para un dado *y* , solo dos fragmentos generados en el paso anterior, así que por cada y pinto entre esos pares.
    - Para interpolar voy a necesitar las coordenadas baricéntricas: las puedo calcular completas para cada fragmento generado, o usar la idea de coherencia (sumar tres $\Delta\alpha_i/\Delta x$ en cada paso de la scanline, y lo mismo habría hecho en DDA).

Sobre la interpolación

- Las coordenadas baricéntricas me dan pesos para una interpolación lineal. Solo el valor de *z* varía en verdad linealmente, para lo demás debería usar hiperbólica (el *z* se usa para corregir los pesos en lugar del *w*, relacionar los α_i con los β_i). Y en el caso de las normales se puede aclarar que ya no serán unitarias (no puedo hacer *slerp* entre 3 vectores) y habrá que renormalizar.

Ej5: Q en Phong

- El exp q controla cuán bien definido se ve el brillo: si es bien puntual, tendiendo a ideal (q muy alto) o está más "desparramado" y bordes menos definidos (q bajo).
- En un material bien pulido, q es alto y las componentes de K_s también.
- Si bajo el q pero mantengo K_s voy a tener más brillo reflejado, porque mantengo el máximo de intensidad y lo "desparramo" en un área más grande. Habríá que bajar K_s para que tenga sentido.

- Si $q = 0$, entonces $cos^q = 1$, así que la componente especular queda constante (como la ambiente, pero en general será blanca e intensa por el K_s, no tiene sentido).

Ej6: Reflejo

- Un modelo de iluminación local, cuando calcula el reflejo en un punto, no tiene en cuenta a los demás objetos (solo a las fuentes de luz directa y el ojo), por eso nunca va a capturar un reflejo.
- Si quiero reflejar un vértice v, según el espejo plano que pasa por el punto p y tiene normal n, la fórmula (según la idea de la ayuda) sería: $f(v) = v - 2((v - p) \cdot n)n$ (si n no está normalizado, dividir por $|n|^2$).
- La transformación es afín. Para armar la matriz que refleje cuaquier cosa, tengo que encontrar el nuevo origen y las nuevas bases para usarlos como columnas de la matriz.
  - Encuentro el nuevo origen transformando el origen original $(0, 0, 0)$ con la función $f(p)$.
  - Encuentro las nuevas bases también transformando las originales con la función $f(p)$, pero como la función transforma puntos, no vectores, planteo las bases como diferencia de vectores. Ej, el eje x era $(1, 0, 0) - (0, 0, 0)$, entonces el transformado será $f(1, 0, 0) - f(0, 0, 0)$.
- Como esta transformación "invierte" las cosas (es fácil de ver que se invierte un solo eje si la normal se alinea con el mismo), entonces cambianos el "giro" del sistema (si era mano derecha, ahora será izquierda), o dicho de otro modo, se invierten las normales (si antes apuntaban hacia "afuera" del objeto, ahora lo harán hacia "adentro").
  - Sabiendo que aquí pasará siempre, basta con invertir (multiplicar por -1) las normales luego de transformarlas (con la transpuestas de la inversa como simpre).
  - En general, si quiero detectar esta situación, veo si el determinante de la matriz es negativo (o más gráficamente, puedo comprobar si con los ejes transformados $e_x \times e_y = e_z$ o $e_x \times e_y = -e_z$).

Errores comunes

Algunos errores observados en muchos de los parciales corregidos:

Ejercicio 2

- Definir vector en lugar de vértice (leer bien!)
- Olvidarse de que vértice y fragmento tienen información asociada además de su posición (como una normal, el color o material para iluminación, las coordenadas textura, o lo que quieran agregar).
- Atar la definición de pixel al monitor (hablan de pantalla y lucecitas). Pixel aplica para cualquier imagen raster, sea o no para mostrar en el monitor

Ejercicio 3

- Pensarlo en 2D o (peor aún) con el plano y la normal alineados con los ejes x e y. El ejercicio empieza diciendo "para el cálculo de iluminación...", así que el vértice puede estar en cualquier lado, en 3D, y la normal apuntando en cualquier dirección.

Ejercicio 4

- Al ir por scanlines, plantear que "pinta" los bordes con DDA, y luego plantear un test del rayo genérico y sin relacionarlo con dda
    - Con dda no pinto, sino que guardo para cada *y* qué fragmentos generé, y uso eso para pintar en la scanline (no tengo que de verdad generar el rayo ni calcular las intersecciones)
- No considerar el efecto de la perspectiva en la interpolación (hiperbólica vs. lineal).

Ejercicio 6

- Al igual que en 3, pensar que el espejo está alineado con los ejes e invertir en la matriz un solo eje.
- Muchos dijeron explícitamente que esto involucraba una rotación de 180°. Un espejado no es una rotación, intenten rotar cualquier cosa no simétrica y nunca van a llegar a la versión espejada.