Aliasing: Frecuencia de muestreo insuficiente, por lo que no se puede recuperar la señal original (teorema del muestreo perfecto de Nyquist). Los efectos del aliasing: las fronteras de las primitivas aparecen escalonadas, falta de continuidad en objetos lejanos. Son típicos los problemas con sombras y texturas.

Antialiasing:

* Supersampling: Partimos de una imagen con mayor resolución y reducimos su tamaño combinando la información de la imagen de mayor resolución.
* Multisampling: Se repite la etapa de rasterizado desplazando la rejilla, y se fusionan los resultados.
* Filtros morfológicos.

Los métodos de iluminación local suelen utilizar modelos heurísticos, basados en el modelo de iluminación de Phong, mientras que los métodos globales suelen estar basados en modelos físicos de la luz.

Clasificación de las fuentes de luz:

* Luz ambiental: Luz residual de la escena sin dirección.
* Luces puntuales: Luces sin dimensión que emiten luz radial en todas las direcciones.
* Luces de área: Luces con 1 o más dimensiones.
* Luces direccionales: Luces situadas en el infinito, cuyos haces de Luz son paralelos. Suelen modelar luces muy lejanas, como la del sol.
* Luz focal: Luz situada en el espacio con una dirección principal. Se puede definir una zona de penumbra.

Propiedades del punto que se está iluminando:

* Propiedades geométricas: Posición, normal, tangente…
* Material: Albedo, ratio de luz especular reflejado…

El modelo de iluminación define la ecuación, mientras que el modelo de sombreado define cómo se aplica la ecuación.

Modelos de iluminación para tiempo real:

* Modelo de iluminación de Phong: Es un modelo heurístico simplificado para iluminar puntos de una escena. En este modelo los objetos no emiten luz, solo reflejan la luz que les llega de las fuentes de luz. Divide el cálculo de luz en 3 componentes:
  + Componente ambiental: Proviene de todas las direcciones e ilumina todas las caras del objeto por igual. Se suele modelar como una constante, evitando que las zonas sin luz directa se visualicen totalmente en negro.
  + Componente difusa: Proviene de una dirección y se refleja en todas direcciones. Podemos atenuar la luz en función de la distancia.
  + Componente especular: Proviene de una dirección y se refleja en una dirección. Modela el brillo de los objetos
  + Iluminación emisiva: Simula objetos que emiten luz, suponiendo que la intensidad que la luz emitida es mayor a la intensidad de la luz que reflejan.

Modelos de sombreado para tiempo real:

* Sombreado plano: Todos los puntos de un polígono se sombrean con el mismo color.
* Gouraud: La intensidad del color se calcula en los vértices y se interpola en los puntos internos.
* Phong: Se interpolan las propiedades de los puntos interiores a la primitiva   
  (normal y las propiedades del color) y se aplica la fórmula de Phong para calcular la intensidad del color en dichos puntos.

Iluminación constante o plana: Se calcula la intensidad del color en uno de los puntos de la primitiva y se utiliza ese color para todos los puntos de esa primitiva. Se puede implementar a nivel de vértices.

Sombreado de Gouraud: Calcula la intensidad del color en los vértices de la malla e interpola dicho color en los puntos internos. Se calcula en la etapa de vértices. Presenta varios problemas, como que la máxima intensidad solo se puede alcanzar en los vértices, se pueden remarcar las aristas, y genera problemas con la reflexión especular.

Sombreado de Phong: A partir de las normales almacenadas en los vértices y de las propiedades del material, esta técnica interpola la normal y las propiedades del material para cada uno de los puntos del polígono. Posteriormente se aplica el modelo de iluminación a dichos puntos. Se implementa en el cauce gráfico pasando desde la etapa de vértices la normal y la información necesaria para obtener las propiedades del material a la etapa de rasterizado, que interpolará dichos atributos para cada fragmento, y después la etapa de fragmentos calcula el color final. Las desventajas de este modelo son que tiene un mayor coste computacional y que hay que pasar más información al shader de fragmentos.

Textura: Una textura es una imagen 2D que representa una propiedad de una superficie.

Mapa de texturas: Estructura de datos que se mapea sobre otra estructura, tradicionalmente una superficie. Generalmente contiene detalles de baja escala.

En el pipeline gráfico, durante la etapa de rasterizado se calculan las coordenadas de textura de los fragmentos, y se calcula la información para que se pueda acceder al nivel de mipmap adecuado. En la etapa de sombreado se accede a las texturasd donde se almacenan información adicional de los fragmentos.

Mapas difusos: Definen cómo el objeto refleja la luz difusa. El albedo es el porcentaje de intensidad de luz reflejado por una superficie.

Mapas especulares: Definen propiedades del objeto sobre cómo refleja la luz especular.

Parallax mapping vs bump mapping: El parallax mapping presenta auto-oclusión en función del punto de vista del usuario.

Mapeado UV: A cada vértice se le asigna una coordenada de textura, y el mapeado se genera de forma explícita. Se suelen utilizar coordenadas normalizadas en texturas cuadradas.

Aliasing en texturas: Pueden haber discontinuidades e información incorrecta en los texeles. Se puede solucionar con los métodos de antialiasing supersampling, pre-filtering, mipmaps, filtrados bilineal (interpolación con 4 téxeles) y trilineal (interpolado del filtrado bilineal entre los niveles de mipmaps más cercanos) y filtrado anisotrópico (se tiene en cuenta la orientación de la primitiva a la hora de generar la textura).

Texturas en opengl: Opengl define las texturas como una colección de una o más imágenes que comparten el mismo formato. Una imagen es un array n-dimensional de píxels (texels) con un formato determinado (especiales, de profundidad…). Pueden ser elementos tanto de entrada, accesibles desde los shaders, como de salida, como bufferes para el renderizado.

Los parámetros que definen una textura: Tipo, tamaño, formato, y parámetros de muestreo.

Se crean con void glGenTextures(GLsizei n, GLuint \* textures);

Se activan con glBindTexture(GLenum target, GLuint texture);

Se borran con glDeleteTexture(GLenum n, Gluint \*texture);

Las texturas son accesibles por los distintos shaders a través de variables uniformes de tipo sampler. Antes de renderizar hay que indicar al shader con qué textura se debe asociar cada sampler. El siguiente paso es indicar las coordenadas de textura, que se deben enlazar como atributos definidos por el usuario. Es necesario usar VAOs.

Deferred shading: Se separa el renderizado en 2 etapas:

* En la primera etapa se calcula la información relativa a los píxeles visibles en la imagen final.
  + El shader de fragmentos no sombrea los distintos píxeles.
  + En su lugar, almacena las propiedades del punto que está procesando en distintas texturas. Las propiedades se utilizarán posteriormente para iluminarlos.
  + El conjunto de estas texturas recibe el nombre de g-buffer.
* Iluminación de los píxeles que componen la imagen final.
  + Esta fase realiza una pasada por cada luz de la escena.
  + En cada pasada se calcula el color de cada píxel teniendo en cuenta solo la fuente de luz que se está procesando.
  + Este color se suma a la imagen final.

G- buffer: Está compuesto por un buffer con las normales, un buffer con las propiedades del material, y el z-buffer vs posición. El número de buffers a utilizar es una decisión de diseño.

Ventajas del deferred shading: Se separa el procesamiento geométrico y las etapas de visibilidad de la iluminación.

Limitaciones del deferred shading:

* No puede mostrar transparencias.
* Cuantas más propiedades se quieran almacenar, mayor tamaño tendrá el g-buffer.
* Problemas con el anti-aliasing en el hardware clásico.

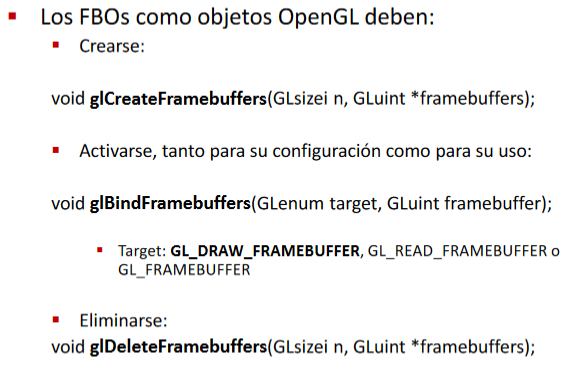
Para implementar deferred shading necesitamos crear texturas utilizando FBOs para poder crearlas en tiempo real. También necesitamos mecanismos que nos permitan renderizar en varios buffers a la vez. El renderizado en múltiples pasadas necesita poder combinar la información del dato antiguo con el dato nuevo (blending).

Implementación de deferred shading en opengl:

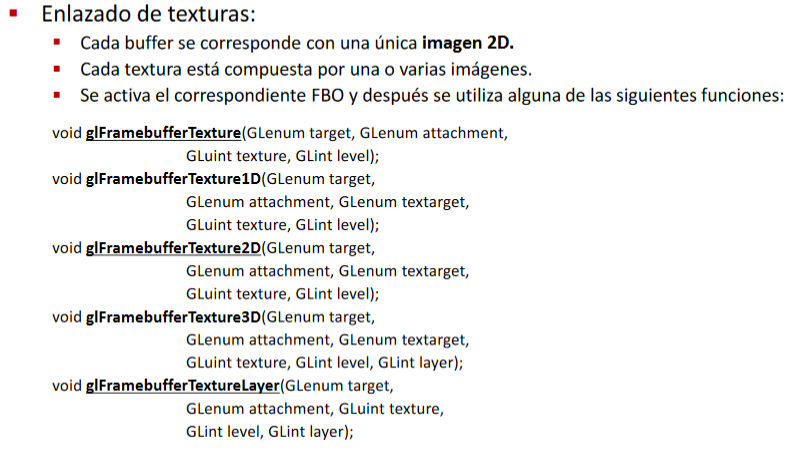
1. Primera pasada:
   1. Se desactiva el blending.
   2. Se activa el buffer de profundidad.
   3. Se activa el FBO que contiene el g-buffer.
   4. Se vacía el g-buffer (clear).
   5. Por cada objeto:
      1. Se configura el shader adecuado con los parámetros del objeto.
      2. Se pinta el objeto.
2. Segunda pasada:
   1. Se desactiva el buffer de profundidad.
   2. Se activa el default framebuffer.
   3. Se vacía el default framebuffer.
   4. Se activa el blending.
   5. Por cada luz:
      1. Se configura el shader correspondiente.
      2. Se pinta un cuadrado con las dimensiones del framebuffer.
   6. Se realiza la operación de cambio de buffers (swapbuffer).

Default framebuffer: Es el conjunto de buffers por defecto que proporciona opengl para renderizar la imagen que se visualizará en pantalla. Está compuesto de los buffers de color, de profundidad, el stencil, el multisampling…

Framebuffer objects (FBOs): Son uno de los mecanismos que tiene opengl para pintar en buffers distintos a los que ofrece opengl por defecto.Permiten renderizar directamente sobre una textura, copiar un framebuffer sobre otro, y es la única forma de escribir en un renderbuffer.



Para poder enlazar una textura o un renderbuffer con un buffer del FBO es necesario que el formato sea compatible.



Renderbuffers: Son objetos que contienen imágenes, al igual que las texturas. Al contrario que estas, están optimizados para bufferes de renderizado.

Blending: Es la etapa del pipeline gráfico que permite fusionar la información de color del fragmento actual con l ainformación que está almacenada en los buffers de color. Por defecto está desactivado, y se activa cuando se quiere utilizar.

Opengl no interactúa con el sistema operativo, sino que deberemos utilizar otras librerías auxiliares para esto.

Shader: Programa que se ejecuta en el servidor y que implementa una etapa del cauce.

En GLSL, el cliente (la CPU) compila en tiempo de ejecución los shaders que va a utilizar. El driver de la tarjeta gráfica dispone de un compilador.

Cauce de opengl3:

* Etapa de vértices (programable).
  + Entrada: atributos (color, posición, normales, coordenadas de textura…).
  + Salida: Posición proyectada (variable especial gl\_Position) y variables variantes (propiedades de los vértices transformadas o creadas).
* Ensamblado.
  + Cambio en el tipo de datos procesado. Agrupa vértices en puntos, líneas, triángulos…
* Etapa geométrica (opcional y programable).
  + Entrada: primitivas (variables variantes de la etapa de vértices + información de adyacencia…).
  + Salida: nuevas primitivas (posición proyectada de los vértices de las nuevas primitivas + variables variantes d ellos vértices de dichas primitivas).
* Clipping y Culling.
* Rasterizado.
  + Transforma la primitiva a espacio de pantalla. Crea fragmentos.
  + Interpola las propiedades calculadas en la etapa de vértices o geométrica (variables variantes) para cada fragmento.
  + Cambio de tipo de datos.
* Etapa de fragmentos (programable).
  + Entrada: variantes interpolados en la etapa de rasterizado.
  + Salida: Color y profundidad.
* Operaciones por fragmento: Z-buffer, blending…
* Operaciones con el framebuffer.

Un shader define un programa para un conjunto de entradas. Las instancias de un shader de un tipo son independientes entre sí.

Etapas programables:

* Vértices:
  + El shader se ejecuta una vez por cada vértice y los procesa.
* Teselación (opcional).
  + Refina la geometría en el proceso de renderizado, y los grupos de vértices se subdividen en primitivas más pequeñas.
* Geométrica (opcional).
  + El shader se ejecuta una vez por primitiva y las procesa.
  + Posee información topológica.
* Fragmentos:
  + El shader se ejecuta una vez por fragmento y calcula el color final de estos.
* Cómputo (opcional).

Variables accesibles desde los shaders:

* Atributos:
  + Propiedades de los vértices (posición, color, coordenadas de textura…).
* Variables uniformes:
  + Parámetros de solo lectura que se mantienen constantes para todos los hilos y en todas las etapas programables (texturas, matriz modelview, propiedades de las luces…).
* Variables variantes:
  + Variables de salida del shader de vértices y entrada del shader de fragmentos (normales).
* Variables especiales predefinidas:
  + Variables propias de cada etapa con una función especial.
* Variables locales:
  + Definidas por el desarrollador.
* Variables de píxeles concretos del framebuffer:
  + Salidas del shader de fragmentos. Definidas por el desarrollador.

Contexto de opengl: El contexto almacena un estado asociado a un dispositivo. Parte de la información se almacena en el cliente (CPU) y parte en el servidor (GPU). Los elementos del contexto son la superficie de renderizado (1 por contexto, el DFB), la configuración del pipeline, los objetos, y los objetos activos.

Tiene que haber un contexto activo antes de utilizar cualquier comando de opengl. El proceso de creación del contexto es independiente del sistema operativo. Un proceso puede tener asociados múltiples contextos, y los distintos contextos de un proceso pueden compartir información.

Las extensiones de opengl sirven para añadir funcionalidades. Existen extensiones pertenecientes al core del propio opengl, y otras que implementa cada fabricante para aprovechar mejor su hardware.

Inicialización de la etapa de aplicación:

* Inicialización:
  + Inicialización del contexto + inicialización del FB.
  + Creación de la ventana.
  + Gestión de las extensiones.
  + Compilación de los shaders.
  + Configuración inicial del cauce.
  + Definición de la geometría que se va a pintar.
* Tratamiento de eventos.
* Renderizado.
  + Establecer los shaders.
  + Configurar los shaders.
  + Configurar etapas no programables.
  + Pintar geometría.
* Liberar recursos.

Se utilizan librerías auxiliares porque opengl no puede llevar algunas tareas a cabo.

FreeGlut se utiliza para crear el contexto, el DFB, inicializar la ventana de renderizado, y registrar las callbacks.

Glew se usa para gestionar las extensiones disponibles, siempre después de crear el contexto.

Framebuffer: Es el conjunto de bufferes sobre los que se renderiza la imagen. Almacena las propiedades de los píxeles de la imagen.

Inicialización de shaders: Los shaders deben ser compilados y enlazados. En la compilación se crea el shader asociado a una etapa específica, y en el enlazado se crea un programa compuesto por shaders de distintas etapas que han de trabajar de manera conjunta.

VBO (Vertex buffer object): Es un buffer que se crea en el espacio de memoria del servidor, que se puede tanto leer como modificar. Los VBOs se almacenan en VAOs (vertex array object).

Para configurar la geometría se sube la información a los VBOs.

Para pintar la geometría se activa el VAO que contiene la información de los VBOs que se van a utilizar. Se pueden pintar mediante un modelo indexado o no indexado.