

Preguntas-Examen.pdf



pablo1mc315



Informática Gráfica



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

BBVA**1/6**

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

Ábrete la Cuenta Online de BBVA y llévate 1 año de **Wuolah PRO**

Ventajas Cuenta Online de BBVA

0€

Sin comisión de administración o mantenimiento de cuenta.
(0 % TIN 0 % TAE)

0€

Sin comisión por emisión y mantenimiento de Tarjeta Aqua débito.

0

Sin necesidad de domiciliar nómina o recibos.

Las ventajas de **WUOLAH PRO**



Di adiós a la publi en los apuntes y en la web



Descarga carpetas completas de un tirón



Acumula tickets para los sorteos

cómo??





1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

ventajas

PRO



Di adiós a la publi en los apuntes y en la web



Acumula tickets para los sorteos



Descarga carpetas completas

estudia sin publi

WUOLAH PRO

RECOPILACIÓN TEORIA IG

1. Explique el modelo de iluminación que se usa en OpenGL (2ptos).

El modelo de iluminación de OpenGL tiene tres componentes:

Componente difusa:

Esta componente modela la reflexión de objetos que no son brillantes, como por ejemplo una pared de yeso o la superficie de un árbol. La reflexión depende de la orientación del objeto o superficie del objeto, por ejemplo, un triángulo, y de la posición de la fuente de luz. La orientación del objeto se modela con el vector normal. La posición de la luz se modela con el vector entre la posición de la luz y un punto del objeto. La reflexión depende del ángulo que forman ambos vectores. Si ambos vectores están normalizados, el ángulo se calcula con el producto escalar.

Además, hay que tener en cuenta una constante de reflexividad difusa del objeto y la componente difusa de la luz.

Componente especular:

Esta componente modela la reflexión de objetos que son brillantes como por ejemplo espejo. La reflexión depende de la orientación del objeto o superficie del objeto, por ejemplo, un triángulo, y de la posición de la fuente de luz. También de la posición del observador. La orientación del objeto se modela con el vector normal. La posición de la luz se modela con el vector entre la posición de la luz y un punto del objeto. Dado el rayo de luz que incide con un ángulo Alfa con respecto a la normal, se produce un rayo reflejado con el mismo ángulo Alfa de salida. Además, tenemos el vector que se forma entre la posición del observador y el punto de incidencia del rayo. La reflexión depende del ángulo que forman el vector del rayo reflejado y el vector del observador. Si ambos vectores están normalizados, el ángulo se calcula con el producto escalar. Además, para modelar la variación de la reflexión especular, se exponencia el producto escalar.

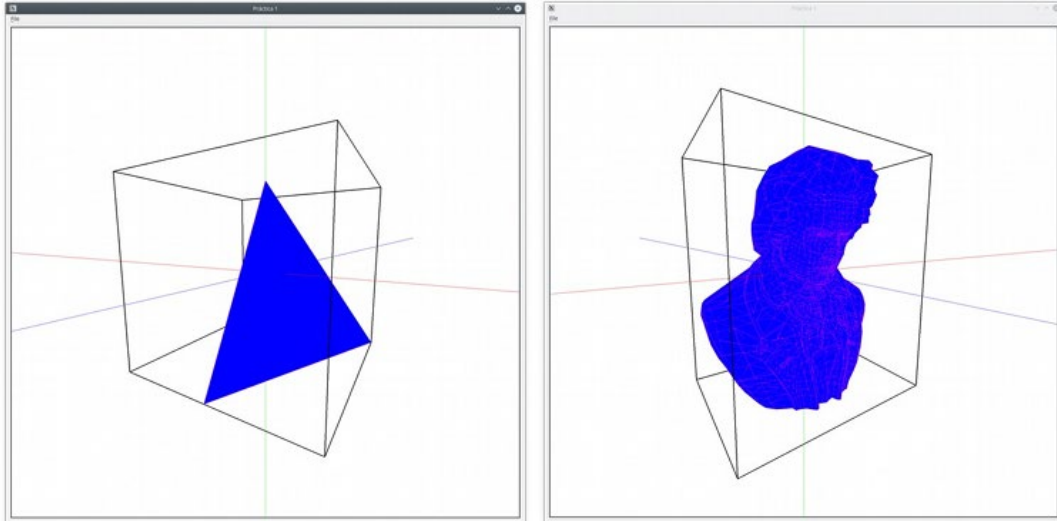
Además, hay que tener en cuenta una constante de reflexividad especular del objeto y la componente especular de la luz.

Componente ambiental:

Esta componente modela la interreflexión de la luz en los múltiples objetos de la escena. Esto implica que la luz reflejada viene de todas direcciones. Se crea para hacer que la parte de los objetos a los que no le llega la luz directamente no aparezca de color negro.

Además, hay que tener en cuenta una constante de reflexividad especular del objeto y la componente especular de la luz.

2. Dado un objeto 3D que se define con definido por sus vértices, `vector<_vertex3f> Vertices`, y triángulos, `vector<_vertex3ui> Triangles`, implemente mediante pseudocódigo o C++ el programa que calcularía la caja frontera (la caja frontera es el menor paralelepípedo que incluye a la figura; basta con salvar el vértice con los coordenadas menores y el vértice con las mayores coordenadas) (2ptos).



```
Pos_min=_vertex3f(1e8,1e8,1e8);
Pos_max=_vertex3f(-1e8,-1e8,-1e8);

for (unsigned int i=0;i<Vertices.size();i++)
{
    If (Vertices[i].x<Pos_min.x)
        Pos_min.x=Vertices[i].x;
    If (Vertices[i].y<Pos_min.y)
        Pos_min.y=Vertices[i].y;
    If (Vertices[i].z<Pos_min.z)
        Pos_min.z=Vertices[i].z;

    If (Vertices[i].x>Pos_max.x)
        Pos_max.x=Vertices[i].x;
    If (Vertices[i].y>Pos_max.y)
        Pos_max.y=Vertices[i].y;
    if (Vertices[i].z>Pos_max.z)
        Pos_max.z=Vertices[i].z;
}
```

Ábrete la Cuenta Online de BBVA y llévate 1 año de Wuolah PRO

Cómo??



Las ventajas de **WUOLAH PRO**



Di adiós a la publi en los apuntes y en la web



Descarga carpetas completas de un tirón



Acumula tickets para los sorteos

Ventajas Cuenta Online de BBVA

0€

Sin comisión de administración o mantenimiento de **cuenta**.
(0 % TIN 0 % TAE)

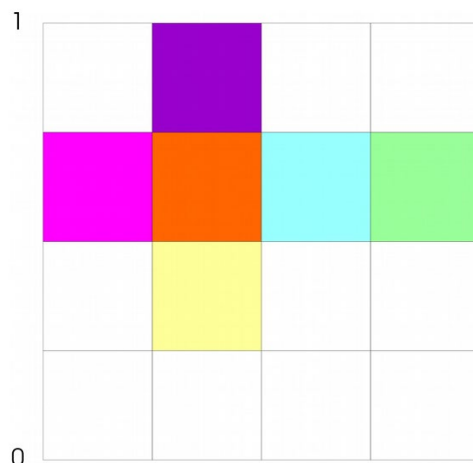
0€

Sin comisión por emisión y mantenimiento de **Tarjeta** Aqua débito.

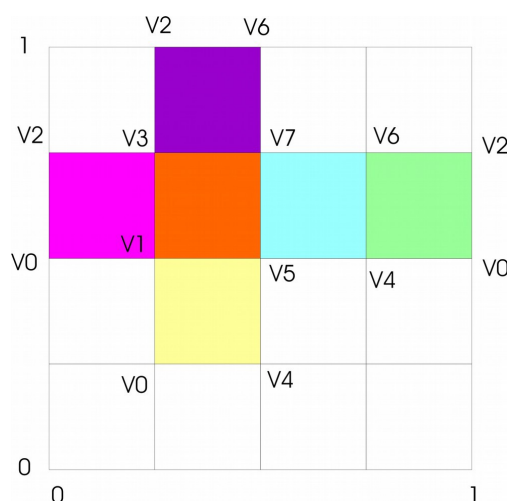
0

Sin necesidad de domiciliar nómina o recibos.

3. Tenemos un cubo definido por sus 8 vértices y 12 triángulos y queremos aplicarle la textura de esta manera (ver imagen). Indicar si se puede hacer o no. En caso de que no se pueda, exponer cómo se resolvería. Indicar los valores de las coordenadas de textura para cada vértice (3ptos).



Si nombramos cada uno de los vértices tal que:



Como se puede ver en la imagen, el problema es que un mismo vértice, y por tanto sus coordenadas de textura, aparecen repetidos. Por ejemplo, el V0 se repite 3 veces. Eso implica que la posición del vértice es la misma, pero para cada cara tiene unas coordenadas de textura.

Por tanto, la solución pasa por repetir los vértices y tener una descripción para los triángulos que usa esos vértices nuevos. Para cada uno de estos nuevos vértices debe haber un elemento que guarde las coordenadas de textura correspondientes.

Por ejemplo, tendríamos

- V0a \rightarrow (0,0.5)
- V0b \rightarrow (0.25,0.25)
- V0c \rightarrow (1,0.5)

De la misma forma habría que poner las coordenadas para cada vértice del cubo.

4. Explique los diferentes métodos que se pueden usar para realizar la selección o pick (2ptos).

Hay tres formas de realizar la selección:

a) Identificación por color

A cada objeto que se quiere identificar se le asigna un identificador, un número natural. Este número es convertido a un color. Se activa la eliminación de partes ocultas. Cuando se dibuja el objeto se usa el color que tiene asociado. Al mover el cursor y pulsar para realizar la selección se guardan las coordenadas x e y del píxel seleccionado. Se lee el píxel del buffer en la posición x e y. Se convierte el color al identificador.

En el caso de representar el color con el modelo RGB y 24 bits, se tienen la posibilidad de identificar $2^{24}-1$ objetos diferentes. El color blanco implica que no se ha seleccionado nada.

Para pasar del identificador al color se usan máscaras de bits para obtener cada parte. Para pasar de color a ID se hacen los pasos inversos.

b) Lanzando un rayo

La idea básica es que cuando pulso el botón del ratón voy a seleccionar el objeto más cercano que está en la posición del cursor. Para ello, obtenemos la posición x e y del cursor en coordenadas de dispositivo y las convertimos a coordenadas de vista. Hacemos pasar una línea recta (o rayo) por el centro de proyección y la nueva posición. Esto es, obtenemos la ecuación de una recta. Calculamos la intersección con los objetos. Si hay intersección se añade a la lista, guardando el identificador del objeto y la profundidad. Por último, ordenamos en profundidad y devolvemos el identificador de la intersección más cercana.

c) Por ventana

Este método consiste en aprovechar la etapa de discretización del cauce visual. Esto es, cuando pasamos de fórmulas (líneas, triángulos, etc) a píxeles. La idea consiste en que cuando se marca la posición con el ratón, se obtiene una posición x e y. Alrededor de dicha posición se crea una pequeña ventana (normalmente unos pocos píxeles). Una vez identificados los píxeles que conforman la ventana, lo único que hay que hacer es dibujar cada objeto, al cual se le asigna un identificador. Si al convertir el objeto en píxeles, coincide con alguno o varios de la ventana, entonces hay selección. Se guarda el identificador del objeto y la profundidad. Finalmente podemos hacer una ordenación por profundidad y quedarnos con el identificador del más cercano.

5. ¿Qué es la transformación de vista?

La transformación de vista es una transformación que permite cambiar de sistema de coordenadas. Esta transformación permite simular el posicionamiento de la cámara en cualquier posición y orientación aplicando transformaciones geométricas (traslaciones, rotaciones, etc.).



1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

ventajas

PRO



Di adiós a la publi
en los apuntes y
en la web



Acumula tickets
para los sorteos



Descarga
carpetas
completas

estudia sin publi
WUOLAH PRO

6. Indique qué parámetros de la cámara están implicados en la transformación de vista y cómo se usan para obtener la transformación de la vista.

VRP: Posición donde está la cámara.

VPN: Hacia donde mira la cámara.

VUP: Indica la orientación hacia arriba.

7. ¿Qué matriz de OpenGL almacena la transformación de vista?

GL_MODELVIEW

8. Cree un ejemplo de transformación de vista incluyendo las llamadas de OpenGL.

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glTranslatef(0,0,-10);  
glRotatef(37,1,0,0);  
glRotatef(45,0,1,10);
```

9. Enumere y explique las propiedades de la transformación de perspectiva.

Acortamiento perspectivo: objetos más lejanos producen una proyección más pequeña.

Puntos de fuga: cualquier par de líneas paralelas convergen en un punto llamado punto de fuga.

Inversión de vista: los puntos que están detrás del centro proyección se proyectan invertidos.

Distorsión topológica: cualquier elemento geométrico que tenga una parte delante y otra detrás del centro proyección produce dos proyecciones semiinfinitas.

10. Queremos realizar acercarnos a un objeto para ver sus detalles. Explicar cómo se podría hacer en una proyección de perspectiva.

La solución más sencilla consiste en acercarse al objeto. El problema está en que si no se cambia el plano delantero habrá un momento en el que se alcanza el objeto y lo recortará. Si colocamos la cámara en una posición donde los planos de corte no recorten el objeto, se puede hacer un zoom simplemente cambiando el tamaño de la ventana de proyección.

11. Dado un cubo definido por sus vértices, `vector<_vertex3f> Vertices`, y sus triángulos, `vector<_vertex3ui> Triangles`, indique lo siguiente si queremos mostrar el cubo texturado:

a) La estructura de datos para guardar las coordenadas de textura

Sería un vector para vértices en dos dimensiones. Su tamaño sería igual al número de vértices:

```
Vector<_vertex2d> Vertices_texcoordinates;  
Vertices_texcoordinates.resize(vertices.size());
```

b) La función que dibujaría el objeto texturado

```
glBegin(GL_TRIANGLES);  
for (unsigned int i=0; i<Triangles.size();i++)  
{  
    glTexCoord2fv((GLfloat *) &Vertices_texcoordinates[Triangles[i]._0]);  
    glVertex3fv((GLfloat *) &Vertices[Triangles[i]._0]);  
    glTexCoord2fv((GLfloat *) &Vertices_texcoordinates[Triangles[i]._1]);  
    glVertex3fv((GLfloat *) &Vertices[Triangles[i]._1]);  
    glTexCoord2fv((GLfloat *) &Vertices_texcoordinates[Triangles[i]._2]);  
    glVertex3fv((GLfloat *) &Vertices[Triangles[i]._2]);  
}  
glEnd();
```

c) Un ejemplo de coordenadas de textura para cada vértice, si la textura se aplica a todas las caras sin repetirla (esto es, cada cuadrado NO muestra la textura completa).

Si no tenemos en cuenta el problema de los puntos repetidos, bastaría con desplegar el cubo sobre la textura y asignar los valores de las coordenadas de textura correspondientes. Por ejemplo:

```
Vertices_texcoordinates[0]=_vertex2f(0,0.5);  
Vertices_texcoordinates[1]=_vertex2f(0.25,0.5);  
Vertices_texcoordinates[2]=_vertex2f(0,0.75);  
Vertices_texcoordinates[3]=_vertex2f(0.25,0.75);  
Vertices_texcoordinates[4]=_vertex2f(0.75,0.5);  
Vertices_texcoordinates[5]=_vertex2f(0.75,0.5);  
Vertices_texcoordinates[6]=_vertex2f(0.5,0.75);  
Vertices_texcoordinates[7]=_vertex2f(0.5,0.5);
```


12. Indica los pasos que hay que realizar en OpenGL y los elementos que intervienen y por tanto han de estar definidos para conseguir que una escena se vea iluminada.

Definir los vectores normales de cada cara: Es necesario definir un vector normal (perpendicular a la superficie apuntando hacia fuera de la parte visible) por cada uno de los vértices de nuestra representación.

Situar las luces: Para iluminar una escena será necesario situar las luces. OpenGL maneja dos tipos de iluminación:

- Luz ambiental: ilumina toda la escena por igual, ya que esta no proviene de una dirección determinada.
- Luz difusa: viene de una dirección específica, y depende de su ángulo de incidencia para iluminar una superficie en mayor o menor medida.

Definiendo materiales: OpenGL permite controlar la forma en que la luz se refleja sobre nuestros objetos, que es lo que se conoce como definición de materiales.

13. Explica el funcionamiento Z-Buffer.

El algoritmo del Z-buffer es del tipo espacio-imagen. Cada vez que se va a renderizar un píxel, comprueba que no se haya dibujado antes en esa posición un píxel que esté más cerca respecto a la cámara (posición en eje Z). Este algoritmo funciona bien para cualquier tipo de objetos: cóncavos, convexos, abiertos y cerrados. Cuando dibujamos un objeto, la profundidad de sus píxeles se guardan en este buffer. Si utilizamos dichos píxeles en pantalla para dibujar otro objeto, se producirá la comparación de las profundidades de dichos píxeles. Si la profundidad del último píxel es mayor que la nueva (está más lejos) el píxel nuevo no se pinta, mientras que si está más cerca (la profundidad es menor), se dibuja el píxel y se guarda la nueva profundidad en el z-buffer. Para activarlo, hay que hacer una llamada a

```
glEnable(GL_DEPTH_TEST)
```

Esta llamada le dice a OpenGL que active el test de profundidad. Además, cada vez que se redibuje la escena, aparte de borrar el buffer de color, hay que borrar el buffer de profundidad. Esto se hace con la llamada

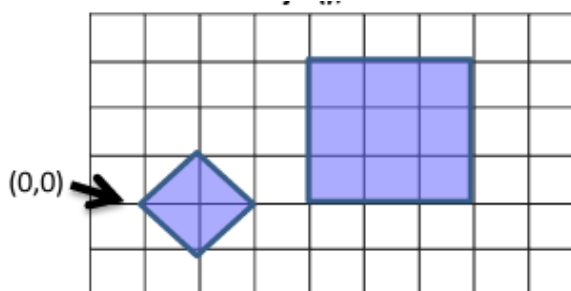
```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT).
```

Por último, pero no menos importante, al inicializar OpenGL se le tiene que definir el buffer de profundidad en el modo de visualización.

14. ¿Qué es una partícula en Informática Gráfica? Describe su ciclo de vida. ¿Puedes poner algún ejemplo?

Las partículas son elementos lógicos a las que se debe otorgar propiedades gráficas para que sean visibles. Por ejemplo, cada partícula se puede sustituir por un objeto geométrico, gracias a lo cual es posible mostrar cualquier forma o dibujo. Con la adición del material correspondiente se pueden utilizar sistemas de partículas para mostrar humo, niebla o fuego.

15. Dada la figura de la izquierda indique qué transformaciones son necesarias para obtener la figura de la derecha. La figura se dibuja con la llamada a la función `cuadrado.dibujar()`.



```
glPushMatrix();
glTranslatef(3, 0, 0);
glScalef(3/√2, 3/√2, 1);
glRotatef(45, 0, 0, 1);
cuadrado.dibujar();
glPopMatrix();
```

16. ¿Cómo se calculan las normales a un vértice?

Podemos definir el vector normal de un vértice como el vector normal de un plano tangente al objeto en dicho vértice, pero esto es muy difícil de calcular. Por ello, realizamos un aproximación muy fiable: la suma de las normales de los triángulos adyacentes a dicho vértice. Para ello, calculamos las normales de todos los triángulos como el producto vectorial de dos de sus aristas, teniendo en cuenta que las normales tienen dirección hacia el exterior del objeto. Las normalizamos con la siguiente fórmula:

$$\frac{n_c}{\|N_c\|}$$

A continuación, sumamos para cada cara, el valor de su normal a la normal de cada uno de sus vértices. Una vez acabado, normalizamos las normales de vértices.

$$\frac{n_v}{\|N_v\|}$$

17. Describa las transformaciones de vista que se pueden aplicar a una cámara.

Las transformaciones de vista son `glFrustum` y `glOrtho`. La declaración de estas transformaciones es:

- `glFrustum (left, right, bottom, top, near, far);`
- `glOrtho (left, right, bottom, top, near, far);`

La función `glFrustum` describe una matriz de perspectiva que produce una proyección en perspectiva.

La función `glOrtho` describe una matriz de perspectiva que produce una proyección paralela.

En ambas, la matriz actual se multiplica por esta matriz y el resultado reemplaza a la matriz actual.



1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

ventajas

PRO



Di adiós a la publi en los apuntes y en la web



Acumula tickets para los sorteos



Descarga carpetas completas

estudia sin publi
WUOLAH PRO

18. Describe brevemente para que sirven, en un programa OpenGL/glut, cada una de estas cuatro funciones:

- glutDisplayFunc(funcion); Esta función se llamará cada vez que se dibuje la ventana.
- glutReshapeFunc(funcion); Función de control del cambio de tamaño de la ventana de visualización.
- glutKeyboardFunc(funcion); Función de control de eventos con el teclado.
- glutSpecialFunc(funcion); Función de control de eventos con el teclado para cuando se ha pulsado una tecla especial.

19. Explique el sistema de colores que se usa en OpenGL.

El sistema de color empleado en OpenGL es el sistema RGB. RGB significa los colores rojo, verde y azul: los colores primarios aditivos. A cada uno de estos colores se le asigna un valor, en OpenGL generalmente un valor entre 0 y 1. El valor 1 significa la mayor cantidad posible de ese color, y 0 significa ninguna cantidad de ese color. Podemos mezclar estos tres colores para obtener una gama completa de colores.

20. Sobre las proyecciones. Indicar si es verdadero V o falso F.

- La proyección de perspectiva acorta los objetos más lejanos (V).
- Dos líneas paralelas en el modelo sólo fugan si no son paralelas al plano de proyección (F).
- El vector Z del sistema cartesiano del observador (punto de mira punto del observador) y el vector de inclinación pueden tener cualquier orientación (V).
- El vector de inclinación siempre coincide con el eje Y del observador (F)
- La ventana debe estar centrada para que se pueda realizar la proyección (F)
- En algunos casos es obligatorio poner el plano delantero detrás del plano trasero (F)
- Si un objeto tiene todos sus vértices detrás del centro de proyección, no se podrá proyectar correctamente (V)
- En una proyección paralela el zoom se puede implementar moviendo los planos de corte (V)

21. Generar el grafo de escena incluyendo las transformaciones, tal que, partiendo de un cubo, un cilindro y un cono unidad, se pueda realizar el modelo de una cámara de TV situada en el borde de una plataforma de radio r. La plataforma (cilindro) rota con respecto al eje Y, la cámara rota toda ella con respecto a su base (cilindro), y permite subir y bajar, y el cuerpo de la cámara (cubo) y el objetivo (cono) rotan arriba abajo e izquierda derecha. Las medidas de las partes del modelo se dejan a discreción. Implementar el modelo en C++ y OpenGL. (nota: se han dibujado los ejes cartesianos como cilindros y no forman parte del modelo) (2.5 pt).

