respuestasexamenteoriagrupoB.pdf

PruebaAlien

Informática Gráfica

3º Grado en Ingeniería Informática

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

1. ¿Qué es la transformación de vista?

   La transformación de vista es una transformación que permite cambiar de sistema de coordenadas, desde el SCM al SCV. Esta transformación permite simular el posicionamiento de la cámara en cualquier posición y orientación. A partir de los parámetros se define el SCV. El SCV se alinea con el SCM aplicando transformaciones geométricas: 1 traslacion y 3 rotaciones.

   ¿Qué parámetros de la cámara están implicados?

   Los parámetros que definen la TV son:

   VRP: Posición donde está la cámara. Origen del SCV. Es un punto dado en el SCM

   VPN: Hacia donde mira la cámara. Es el eje z del SCV. Es un vector dado en el SCM

   VUP: Indica la orientación hacia arriba. Es un vector dado en el SCM

   ¿Qué matriz de OpenGL almacena dicha transformación?

   La GL_MODELVIEW

   Cree un ejemplo incluyendo las llamadas de OpenGL.

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

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

   a) Acortamiento perspectivo: objetos más lejanos producen una proyección más pequeña

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

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

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

3. Queremos realizar acercarnos a un objeto para ver sus detalles. Explicar cómo se podría hacer en una proyección de perspectiva, ventajas e inconvenientes.

   a) 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

   b) 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.

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

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

   Sería un vector para dos flotantes. Su tamaño sería igual al número de vértices

   Vector<_vertex2d> Vertices_texcoordinates;

   2) 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();
```

   3) 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);