Benemérita Universidad Autónoma de Puebla.

EXAMEN DE SEGUNDO PARCIAL.

Graficación.

NOMBRE: Pedro Vargas Arenas

Matrícula: 201734553

INTRODUCCIÓN.

Para concluir el segundo parcial se tiene un proyecto para poner a prueba los conceptos vistos y aprendidos en clase. El proyecto de según parcial consiste en realizar un escenario 3D usando perspectivas y debe estar compuesto por figuras elementales (cubos, pirámides). Se obtiene la matriz de modelado de cada objeto y rotaciones dependiendo del eje que se quiere o aplicando la rotación libre sobre el mismo. Las matrices necesarias para la matriz de modelado son: Traslación, escalamiento, rotaciones sobre los ejes x, y, z y rotación libre. Para obtener la matriz de modelado necesaria, se deben multiplicar las matrices se una determinada manera para obtenerla.

De igual manera, se incluyen técnicas de graficado nativo, ya que las figuras básicas se deben de crear de esta manera, también los objetos compuestos se crean nativamente con las figuras anteriores. También se usa programación orientada a objetos para usar clases, figuras unitarias para simplificar operaciones de la matriz de modelado.

OBJETIVOS.

- Aplicar los conceptos adquiridos sobre los gráficos 3D en OpenGL.
- Implementar las matrices de operaciones (rotación, traslación, escalamiento) en 3D.
- Implementar y crear la matriz de rotación libre.
- Implementar y crear las matrices de rotación sobre los ejes x, y, z.
- Practicar las técnicas de graficado vistas en clase.
- Aplicar los conceptos de perspectivas en el entorno 3D.
- Reutilizar códigos previamente realizados como actividad.
- Aplicar operaciones con matrices y vectores.
- Construcción de elementos compuestos por figuras simples en 3D.
- Practicar las operaciones para llegar a matrices de rotación necesarias.
- Aplicar objetos unitarios para la operación de estos.

MARCO TEÓRICO.

¿Qué es la rotación libre?

En la rotación libre se define un vector en el espacio el cual será la referencia para la rotación del objeto. La diferencia contra otros tipos de rotación es que en este caso no es necesario un eje para rotar, ya que este se define en el espacio y sobre él se trabaja.

Proyección ortogonal.

Una proyección ortogonal define un volumen de la vista de tipo paralelepipédico tal y como se muestra en la siguiente figura. La principal característica de esta proyección es que el tamaño de los objetos es independiente de la distancia a la que estén del observador, por ejemplo, dos cilindros del mismo tamaño, uno a cinco unidades y el otro a diez unidades de distancia del observador se proyectarán con el mismo tamaño.

Matrices de operación (rotación, traslación, escalamiento).

Escalamiento.

El escalamiento permite cambiar el tamaño de un objeto expandiéndolo o contrayéndolo en sus dimensiones.

En 3D implica el cambio de tamaño de un polígono, donde cada punto $p = (x_1,\ x_2,\ x_3)$ es transformado por la multiplicación de dos factores de escalamiento s_1, s_2 y s_3 a lo largo de sus ejes respectivos. Por lo que matriz se define como:

$$\begin{matrix} Ex & 0 & 0 & 0 \\ 0 & Ey & 0 & 0 \\ 0 & 0 & Ez & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Traslación.

La traslación permite desplazar un objeto a lo largo de sus dimensiones, como resultado se obtiene un cambio de posición.

En 3D implica el desplazamiento de un polígono, donde cada punto $p = (x_1,\ x_2,\ x_3)$ es trasladado d_1 unidades en el eje X_1 , d_2 unidades en el eje X_2 y d_3 unidades en el eje x_3, de esta forma, las coordenadas del nuevo punto se obtienen con la siguiente matriz:

$$\begin{matrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{matrix}$$

Rotación.

La rotación permite girar un objeto sobre un eje de rotación, dado un valor de ángulo de rotación θ y su dirección.

En 3D se lleva a cabo alrededor de un punto y sobre un distinto eje, se especifica el ángulo de rotación θ, y el punto de rotación (pivote) sobre el cuál el objeto será rotado. Utilizando coordenadas polares, el punto $p = (x_1,\ x_2,\ x_3)$ se puede escribir como $p = (r,\ \theta)$ y el punto $p' = (x_1',\ x_2',\ x_3')$ como $p' = (r,\ \phi + \theta)$. Su representación matricial es:

Para el eje *x.*

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Para el eje *y*.

$$\begin{matrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Para el eje *z.*

$$\begin{matrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

DESARROLLO Y CONOCIMIENTOS ADQUIRIDOS.

Para desarrollar este proyecto, se definieron las clases Punto y Línea para manipularlas con las matrices de rotación.

Como se necesitan figuras elementales, se crearon las siguientes clases:

- Cubo.
- Círculo.
- Pirámide.
- Triángulo.
- Cuadrado.

Dentro de cada clase, se tienen métodos para asignar los puntos de la figura a un arreglo de tipo Punto y así acceder a estos de una manera más simple. Los puntos que se asignan son los unitarios.

- En clase *Cubo.*

```
void Cubo::cuboUnitario(){

    puntos[0].setValues(0, 0, 0);
    puntos[1].setValues(1, 0, 0);
    puntos[2].setValues(1, 1, 0);
    puntos[3].setValues(0, 1, 0);
    puntos[4].setValues(0, 0, 1);
    puntos[5].setValues(1, 0, 1);
    puntos[6].setValues(1, 1, 1);
    puntos[7].setValues(0, 1, 1);
}
```

- En clase *Círculo.*

```
void Circulo::circuloUnitario(){

    int n = 0;

    for(float i = 0; i < 2*(atan(1)*4); i += 0.128){
        puntos[n].setValues(1*cos(i), 1*sin(i), 0);
        n++;
    }
}
```

- En clase *Pirámide.*

```
void Piramide::piramideUnitario(){

    puntos[0].setValues(0, 0, 0);
    puntos[1].setValues(1, 0, 0);
    puntos[2].setValues(1, 0, 1);
    puntos[3].setValues(0, 0, 1);
    puntos[4].setValues(0.5, 0.5, 0.5);
}
```

- En clase *Triángulo*.

```
void Triangulo::trianguloUnitario(){

    puntos[0].setValues(0, 0, 0);
    puntos[1].setValues(1, 0, 0);
    puntos[2].setValues(0.5, 1, 0);
}
```

- En clase *Cuadrado*.

```
void Cuadrado::cuadradoUnitario(){

    puntos[0].setValues(0, 0, 0);
    puntos[1].setValues(1, 0, 0);
    puntos[2].setValues(1, 1, 0);
    puntos[3].setValues(0, 1, 0);
}
```

Al igual se cuenta con funciones que asignan la matriz de rotación, traslación y escalamiento respectivas, también se tiene una función que asigna el vector para la rotación libre. Para asignar estas matrices a cada figura, se requiere una clase *Matriz* y *MatrizRotaciones* estas se instancian dentro de las clases antes mencionadas para manejar las matrices.

Esto se realiza en todas las clases (incluyendo las clases *Punto* y *Línea*).

```
void Cubo::setPuntosMatriz(float x0, float y0, float z0, float x1, float y1, float z1){

    matriz.setPuntos(x0, y0, z0, x1, y1, z1);
}

void Cubo::setMatriz(float x, float y, float z, float e, float grados){

    matriz.setTraslacion(x, y, z);
    matriz.setEscalamiento(e);
    matriz.matrices(grados);
}
```

La primera función asignar el vector para la rotación libre y la segunda inicia la matriz de traslación, escalamiento y las de rotación.

Como se quieren el *X*, *Y* y *Z* de los puntos de las figuras, estos se guardan en un arreglo de flotante junto con un 1 para su posterior operación con la matriz de modelado.

```
void Cubo::puntosCubo(int i){

    puntosF[0] = puntos[i].getX();
    puntosF[1] = puntos[i].getY();
    puntosF[2] = puntos[i].getZ();
    puntosF[3] = 1;
}
```

Los puntos resultantes se regresan y guardan dentro del mismo arreglo de tipo Punto. Todo este proceso se repite en cada clase exceptuando la clase *Punto*.

```
void Cubo::pCubo(int i){

    puntos[i].setValues(matriz.getPunto(0), matriz.getPunto(1), matriz.getPunto(2));
}
```

Dentro de la clase *Matriz* se definen las matrices para rotar, trasladar y escalar dependiendo de la traslación y escalamiento que se quiera, esto se recibe como parámetro al inicializarlas.

En esta clase, se calcula la matriz de rotación libre. Para esto, se definen las matrices de rotación en *x, y, z* y las inversas de *x* y *y*. Para inicializar las matrices anteriores, se recibe como parámetro el vector de referencia y los puntos se operan para obtener los valores *a, b, c* y *d* que son necesarios para realizar lo anterior.

```
void Matriz::setPuntos(float x0, float y0, float z0, float x1, float y1, float z1){

    float x = (x1 - x0), y = (y1 - y0), z = (z1 - z0);
    float v = sqrt(pow((double)x, 2) + pow((double)y, 2) + pow((double)z, 2));

    a = x/v;
    b = y/v;
    c = z/v;
    d = sqrt(pow((double)b, 2) + pow((double)c, 2));
}
```

Una vez hecho lo anterior, se procede a calcular la matriz de rotación libre multiplicando las matrices de rotación en diferentes puntos de la siguiente manera:

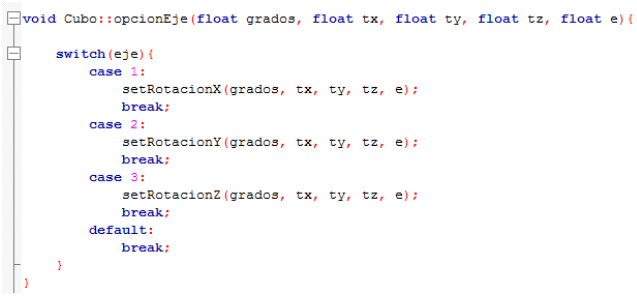
$$(Rx)^{-1} * (Ry)^{-1} * Rz * Ry * Rx$$

Una vez obtenida esta matriz, se multiplica con las matrices de traslación y escalamiento en el siguiente orden:

$$T * R * E$$

Y de esta manera se obtiene la matriz de operaciones con rotación libre. Como la función en donde se realiza todo esto recibe los puntos para su transformación, estos se multiplican por la matriz de operaciones resultante y se regresan para su dibujado.

Ahora, como también se implementó la rotación sobre ejes, se tiene una variable en todas las clases para elegir el eje. Para saber el eje, la variable tiene su respectivo set y get y así cambiar el valor, también se tiene un método que compara el valor de la variable y así elige el método para que rote con el eje.



```
void Cubo::opcionEje(float grados, float tx, float ty, float tz, float e){

    switch(eje){
        case 1:
            setRotacionX(grados, tx, ty, tz, e);
            break;
        case 2:
            setRotacionY(grados, tx, ty, tz, e);
            break;
        case 3:
            setRotacionZ(grados, tx, ty, tz, e);
            break;
        default:
            break;
    }
}
```

Adentro de cada función que se llama, se multiplica la matriz de rotación elegida por la matriz de traslación y escalamiento.

- Para *x.*

$$T * Rx * E$$

- Para *y.*

$$T * Ry * E$$

- Para *z.*

$$T * Rz * E$$

Se realiza el mismo método de la rotación libre para transformar los puntos

Para iniciar con la construcción del escenario, se crearon más clases, las cuales contienen una serie de figuras básicas que están colocadas de tal manera que asemeja la figura que se quiere.

Para lograr esto, se manipulan las matrices de los cuadrados, círculos, triángulos cubos y pirámides a necesitar, es decir, el escalamiento, los grados de rotación y la posición en donde se quiere la figura se mandan como parámetro a la función encargada de generarla.

Las clases que se encargan de esto son:

- Escenario.
- Terrestre.
- Nave.
- Árbol.
- Ciudad.
- Muro.
- Persona.

Para la clase *Terrestre* está compuesta por cubos, pirámides y líneas. La manera en que estas componen al objeto, es un tanto compleja ya que se requirieron muchos puntos para lograrlo. También, la clase contiene un par de métodos que dibujan “rayos” con ayuda de líneas.

En la clase *Nave* crea dos naves, una que implica círculos y pirámides, y la otra nave ocupa pirámides. Debido a que no se logró realizar una esfera, se recurrió a crear un círculo y darle profundidad.

Para la clase *Árbol* se necesitaron un cubo con el cual se formó un prisma cuadrangular que tiene la función del tronco y una pirámide para simular la vegetación del mismo.

La clase *Ciudad* sólo se crearon edificios con ayuda de cubos que igual en la anterior clase, forman prismas cuadrangulares.

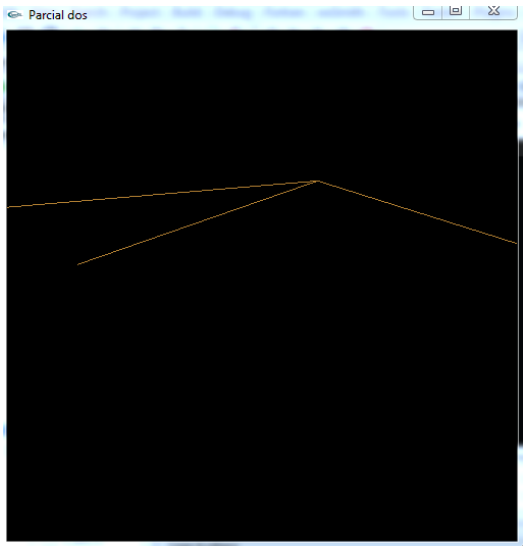
Por último, en la clase *Muro*, se usaron líneas para dibujar una ligera barrera y un cubo para formar una puerta que se escala.

Todo esto se llama en la clase *Escenario* y adentro de esta, se realizan las traslaciones, escalamientos y rotaciones correspondientes para colocar los objetos en el lugar que se quiere.

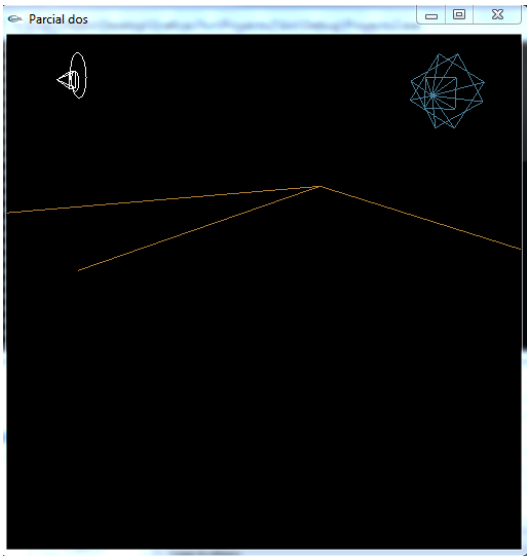
La clase *Persona* consiste en un círculo y cinco líneas para dibujar una persona.

PRUEBAS.

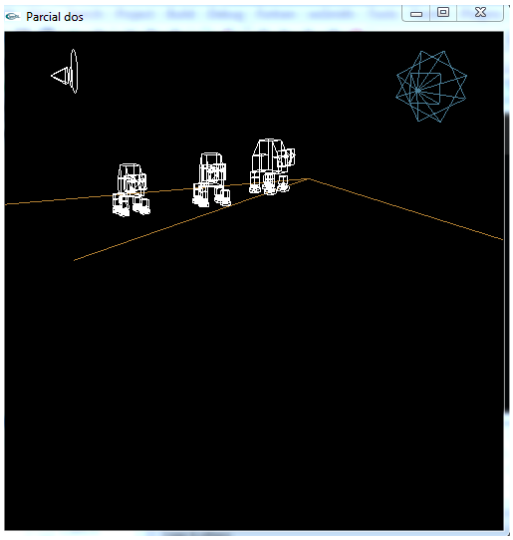
Montaña:



Naves:

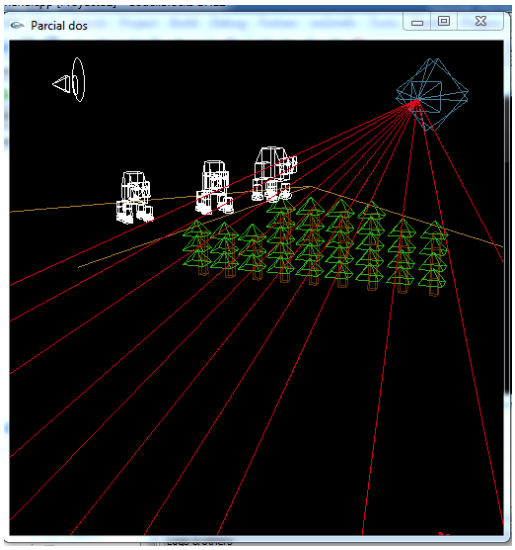


Vehículos terrestres:



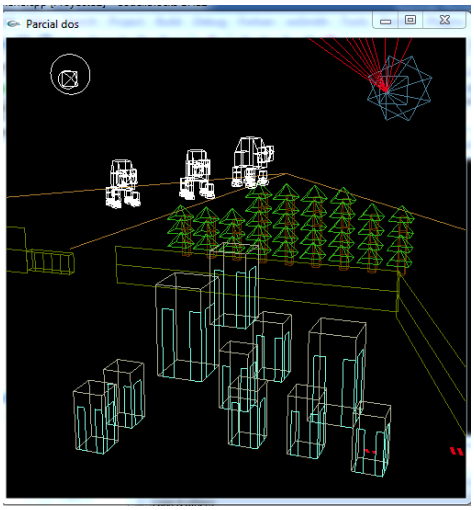
Árboles:

Rayos:

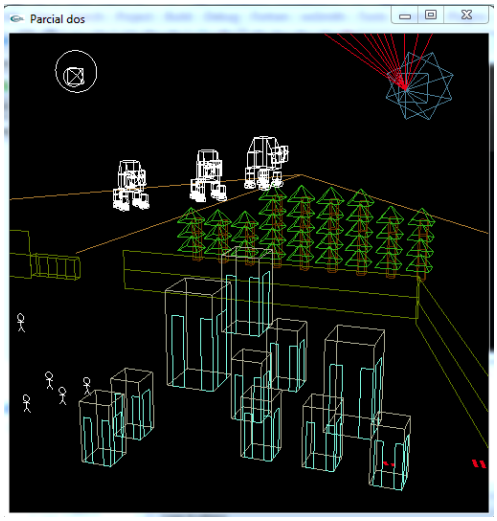


Muros:

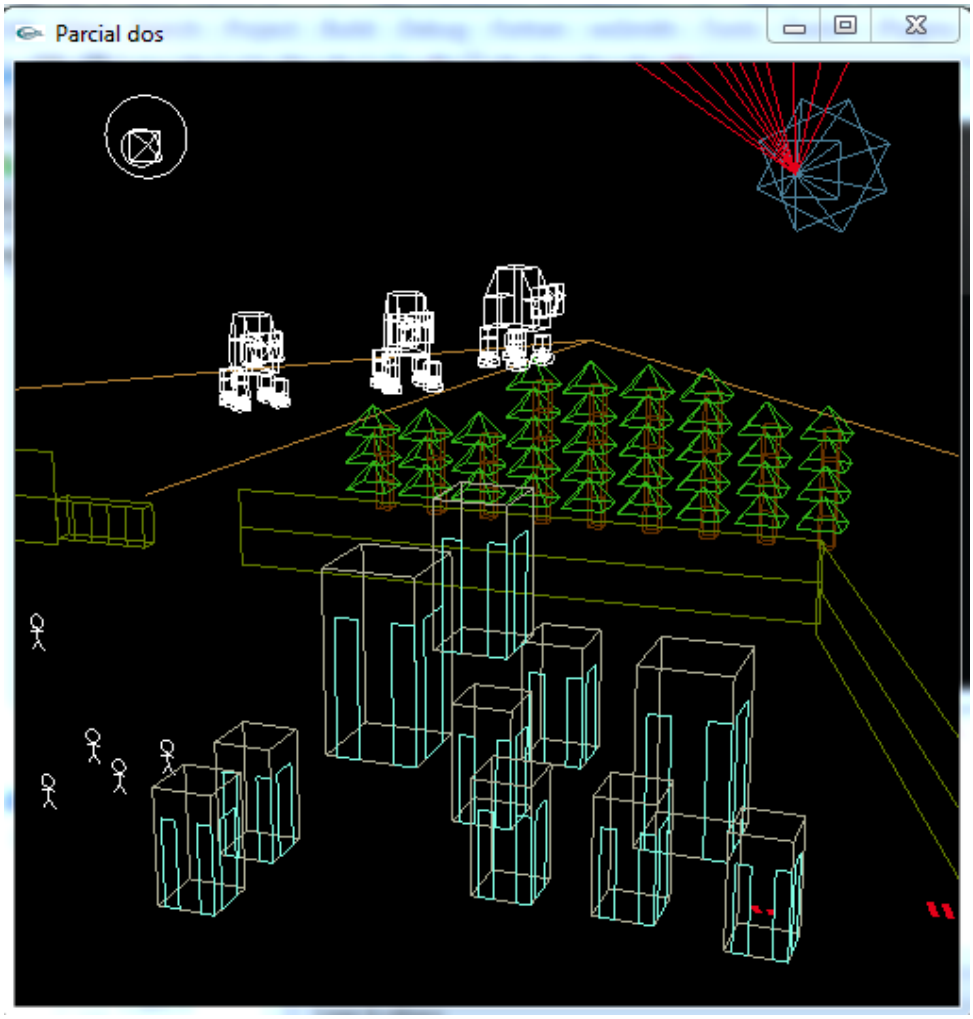
Edificios:



Personas:



RESULTADO FINAL:



CONCLUSIONES.

Como el proyecto exigía cierta dificultad, hubo muchas complicaciones, entre estas, hubo con el manejo de los objetos unitarios, la rotación de los objetos, pero en particular, con el manejo y operación de las matrices tanto para la rotación en un eje como la rotación libre, ya que tenía que realizarse de una sola manera y, además, multiplicarse con los puntos, esto se tenía que hacer con un buen manejo de índices y ciclos, porque es la forma más óptima de operar estas estructuras y gracias a que todo el proyecto está basado en clases, esto tuvo una solución eficaz y sin hacer operaciones más complejas como el manejo de doble apuntador.

Con los conceptos que analizaron durante las clases se facilitó en cierta manera el uso de las clases, cómo manipular las líneas y los puntos que se necesitaban para dibujar, la asignación de los puntos, entre otros. Lo que requirió más esfuerzo fue el dibujado de los objetos compuestos ya que se tuvieron que calcular muchos puntos y además, transformarlos, sin embargo, con ayuda de los conceptos previos a la realización del proyecto, se logró finalizarlo con éxito.

Gracias a la elaboración de este programa se pudo observar la complejidad que hay atrás de este lenguaje y el trabajo que se tiene que hacer con funciones primitivas, además de que se logró una mejor compresión los conceptos en cuanto al dibujado en 3D y cómo se determinan.

BIBLIOGRAFÍA.

https://elbauldelprogramador.com/clases-y-objetos-introduccion/

http://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-3-matrices/

http://catarina.udlap.mx/u_dl_a/tales/documentos/mcc/cruz_m_ia/capitulo3.pdf}

Mathematics for Computer Graphics, Second Edition, John Vince. Springer.

Introduction to Computer Graphics. Frank Klawonn. Springer.