Universidad Nacional Autónoma de México.

Facultad de Ingeniería.

Práctica 5:
Colisiones

Diseño de interfaces, multimedia y realidad virtual.

Grupo 01.

José de Jesús Hernández Castro.

Semestre 2022-1

Modelos de colisión.

El objetivo de la práctica era el de implementar dos de los modelos de colisión más utilizados; Caja y esfera, en OpenGL, y a su vez detectar colisiones entre ellos; Esfera-Esfera, Caja-Caja, Esfera-Caja.

En el proyecto destaca la librería de model.h, la cual contiene la clase y las funciones de carga básicas para modelos obj (práctica 3), además de generar los respectivos modelos de colisión acorde a una bandera que nos indica cual de los 2. Los métodos de generación de los modelos y de detección de colisiones, fueron los vistos en clase durante el semestre.

Al realizar la práctica se logró cumplir totalmente con los objetivos planteados, puesto que los modelos de colisión se generaron sin problemas y la detección de las tres variantes de colisión funcionaron como era de esperarse. Como conclusión adicional, se pudieron observar las ventajas y desventajas de los dos modelos de colisión implementados, por ejemplo, los espacios muertos del modelo de esfera.

Controles de Usuario

Para permitir la interacción con el usuario a través de teclado con la siguiente funcionalidad:

- Teclas **AD** realiza un movimiento lateral relativo a la vista de la cámara (strafe).
- Teclas **WS** para avanzar o retroceder.
- Teclas **WS** + (**LeftShift | RightShift**) para subir o bajar solo en el caso de la cámara libre.
- Tecla **F** para cambiar entre cámaras.
- Clic derecho + desplazamiento del ratón para mover la vista.
- Clic izquierdo para seleccionar alguno de los modelos 3D.
- Teclas de dirección ↑↓←→ para mover el modelo 3D, seleccionado con el ratón, sobre los ejes Z y X respectivamente.
- Teclas de dirección ↑↓ + (**LeftShift | RightShift**) para mover el modelo 3D, seleccionado con el ratón, sobre el eje Y..

Procedimiento

1- Al archivo de cabecera que contiene las definiciones de función para carga de modelos obj (model.h), se le añadieron los atributos relevantes de cada uno de los modelos de colisión.

```
char colision;//tipo de modelo de colisión (caja o esfera)
//Para el modelo de esfera
GLfloat center[3] = { 0,0,0 };
GLfloat radius = 0;
```

```
//Para el modelo de caja
GLfloat maxX=0;
GLfloat maxY=0;
GLfloat maxZ=0;
GLfloat minX = 0;
GLfloat minY = 0;
GLfloat minZ = 0;
```

2- Como siguiente paso, se añadió la generación del modelo, dentro de la función de dibujo sólido del cargador.

```
//Colisiones
   int vn = vertices.size();//Número de vertices del modelo
   if (colision == 's' || colision == 'S') {
       //Colision esfera
       double radio = 0, aux = 0;
       vertex centro = { 0,0,0 };//Nuestro centro en (0,0,0)

       double theta, phi;
       int meridianos = 20, paralelos = 20;

       GLfloat v1[] = { 0,0,0 };
       GLfloat v2[] = { 0,0,0 };
       GLfloat v3[] = { 0,0,0 };
       GLfloat v4[] = { 0,0,0 };
       GLfloat angulom, angulop;
       //Calculo del centro de la esfera
       for (i = 0; i < vn; i++) {
           centro.x += vertices[i].x;
           centro.y += vertices[i].y;
           centro.z += vertices[i].z;
       }
       //Centro final de nuestra esfera
       centro.x = centro.x / vn;
       centro.y = centro.y / vn;
       centro.z = centro.z / vn;

       this->center[0] = centro.x;
       this->center[1] = centro.y;
       this->center[2] = centro.z;

       for (i = 0; i < vn; i++) {
           aux = longitud(centro.x, centro.y, centro.z,
               vertices[i].x, vertices[i].y, vertices[i].z);
           if (aux > radio)
               radio = aux;//Radio final de nuestra esfera
       }
       this->radius = radio;


       //Dibujo de la esfera
       angulom = 2 * 3.141592654 / meridianos;
```

```
    angulop = 3.141592654 / paralelos;

    glDisable(GL_LIGHTING);
    glTranslatef(centro.x, centro.y, centro.z);
    glBegin(GL_LINES);

    for (i = 0; i < meridianos; i++)
    {
        for (j = 0; j < paralelos; j++)
        {
            v1[0] = radio * cos(angulom * i) * sin(angulop * j);
            v1[1] = radio * cos(angulop * j);
            v1[2] = radio * sin(angulom * i) * sin(angulop * j);

            v2[0] = radio * cos(angulom * i) * sin(angulop * (j + 1));
            v2[1] = radio * cos(angulop * (j + 1));
            v2[2] = radio * sin(angulom * i) * sin(angulop * (j + 1));

            v3[0] = radio * cos(angulom * (i + 1)) * sin(angulop * (j + 1));
            v3[1] = radio * cos(angulop * (j + 1));
            v3[2] = radio * sin(angulom * (i + 1)) * sin(angulop * (j + 1));

            v4[0] = radio * cos(angulom * (i + 1)) * sin(angulop * j);
            v4[1] = radio * cos(angulop * j);
            v4[2] = radio * sin(angulom * (i + 1)) * sin(angulop * j);

            glColor3f(1, 0, 0);
            glVertex3fv(v1);
            glVertex3fv(v2);
            glVertex3fv(v3);
            glVertex3fv(v4);
        }
    }
    glEnd();
    glEnable(GL_LIGHTING);

}else if (colision=='b' || colision=='B') {
    GLfloat minX = 65000, minY = 65000, minZ = 65000;
    GLfloat maxX = -65000, maxY = -65000, maxZ = -65000;

    glDisable(GL_LIGHTING);
    glBegin(GL_LINE_STRIP);

    for (i = 0; i < vn; i++) {
        if (vertices[i].x < minX)
            minX = vertices[i].x;//minimo en x

        if (vertices[i].x > maxX)
            maxX = vertices[i].x;//maximo en x

        if (vertices[i].y < minY)
            minY = vertices[i].y;//minimo en y
        if (vertices[i].y > maxY)
```

```
            maxY = vertices[i].y;//maximo en y

        if (vertices[i].z < minZ)
            minZ = vertices[i].z;//minimo en z
        if (vertices[i].z > maxZ)
            maxZ = vertices[i].z;//maximo en z
    }
    this->maxX = maxX; this->maxY = maxY; this->maxZ = maxZ;
    this->minX = minX; this->minY = minY; this->minZ = minZ;
    GLfloat vertice[8][3] = {
            {minX,minY, minZ},
            {minX,minY, maxZ},
            {minX,maxY, maxZ},
            {minX ,maxY, minZ},
            {maxX,minY, minZ},
            {maxX,minY, maxZ},
            {maxX,maxY, maxZ},
            {maxX ,maxY, minZ},
    };
    //Dibujo de la caja
    glVertex3fv(vertice[0]);
    glVertex3fv(vertice[1]);
    glVertex3fv(vertice[2]);
    glVertex3fv(vertice[3]);
    glVertex3fv(vertice[7]);
    glVertex3fv(vertice[4]);
    glVertex3fv(vertice[5]);
    glVertex3fv(vertice[6]);
    glVertex3fv(vertice[2]);
    glVertex3fv(vertice[1]);
    glVertex3fv(vertice[5]);
    glVertex3fv(vertice[4]);
    glVertex3fv(vertice[0]);
    glVertex3fv(vertice[3]);
    glVertex3fv(vertice[7]);
    glVertex3fv(vertice[6]);

    glEnd();
    glEnable(GL_LIGHTING);
    }
    else {
    printf("\nModelo sin colisión\n");
    }
```

3- En el programa principal, declaramos todas las variables necesarias para el manejo de los modelos 3D, esto es debido a que se tiene que considerar tanto la escala como la posición de cada uno de ellos respecto a sus propias variables de modelo de colisión (por ejemplo, el desplazamiento del centro de la esfera o la escala del radio).

```
obj3dmodel corvette;
obj3dmodel goku;
obj3dmodel test;
```

```
obj3dmodel test2;
//Modelo de prueba para mostrar cómo es que se cargan y dibujan los modelos que no están
formados por triángulos
obj3dmodel tie;
//Variables de posición de modelos
GLfloat posMono[3] = { -10, 10, 40 };
GLfloat posCorvette[3] = { -10, 50, 0 };
GLfloat posGoku[3] = { -60,10,0 };
GLfloat posTest2[3] = { -60,10,40 };
//Variables de escala de modelos
GLfloat scaleMono = 3;
GLfloat scaleCorvette = 3;
GLfloat scaleGoku = 3;
GLfloat scaleTest2 = 3;
//Variables de selección de modelo
bool selNave = false;
bool selGoku = false;
bool selMono = false;
bool selTest2 = false;
//Variables de coordenadas de picking
GLdouble relX, relY, relZ;
//Distancias
GLfloat distGoku,distCorv,distTest,distTest2;
```

4- Ahora declaramos una serie de funciones que nos ayudarán a realizar ciertas comparaciones y cálculos necesarios para detectar colisiones, como por ejemplo las distancias respecto a los centros de las esferas, o los máximos y mínimos de las cajas.

```
//Funciones sobrecargadas de distancia
GLfloat distancia(obj3dmodel model1, obj3dmodel model2,GLfloat *pos1, GLfloat *pos2) {
    //Distancia entre centros de esferas
    return sqrt(pow(((model1.center[0]+pos1[0]) - (model2.center[0]+ pos2[0])), 2) +
                pow(((model1.center[1]+pos1[1]) - (model2.center[1]+ pos2[1])), 2) +
                pow(((model1.center[2]+pos1[2]) - (model2.center[2]+ pos2[2])), 2));
}

GLfloat distancia(obj3dmodel model, GLdouble x, GLdouble y, GLdouble z,GLfloat *pos){
    //Distancia entre el punto picking y el centro de la esfera
    return sqrt(pow((x - (model.center[0] + pos[0])), 2) +
        pow((y - (model.center[1] + pos[1])), 2) +
        pow((z - (model.center[2] + pos[2])), 2));
}
//Esta función sirve para probar si un punto está dentro de una caja (picking)
bool puntoEnCaja(obj3dmodel model, GLdouble x, GLdouble y, GLdouble z, GLfloat
scale,GLfloat *pos) {

    if (model.maxX * scale + pos[0] > x && model.maxY * scale + pos[1] > y && model.maxZ
* scale + pos[2] > z )
        if(model.minX * scale + pos[0] < x && model.minY * scale + pos[1] < y &&
model.minZ * scale + pos[2] < z)
        return true;
```

```
    else
        return false;
}
//Esta función sirve para probar la colisión de dos cajas
bool cajaEnCaja(obj3dmodel model1, obj3dmodel model2, GLfloat scale1, GLfloat* pos1,
GLfloat scale2, GLfloat *pos2) {
    bool x, y, z;

    //De todos los casos posibles entre que haya o no colisión de cajas, se decidió
evaluar los más sencillos
    //los cuales son en donde no hay colisión, caso contrario si hay colisión

    //eje x
    if ((model2.maxX * scale2 + pos2[0] < model1.minX * scale1 + pos1[0]) ||
        (model1.maxX * scale1 + pos1[0] < model2.minX * scale2 + pos2[0]))
        x = false;
    else
        x = true;
    //eje y
    if ((model2.maxY * scale2 + pos2[1] < model1.minY * scale1 + pos1[1]) ||
        (model1.maxY * scale1 + pos1[1] < model2.minY * scale2 + pos2[1]))
        y = false;
    else
        y = true;
    //eje z
    if ((model2.maxZ * scale2 + pos2[2] < model1.minZ * scale1 + pos1[2]) ||
        (model1.maxZ * scale1 + pos1[2] < model2.minZ * scale2 + pos2[2]))
        z = false;
    else
        z = true;

    return x && y && z;
}
//Esta función sirve para probar la colisión esfera-caja
bool esferaEnCaja(obj3dmodel model1, obj3dmodel model2, GLfloat scale1, GLfloat* pos1,
GLfloat scale2, GLfloat* pos2) {
    bool x, y, z;
    GLfloat d = 0;

    //eje x
    if (model1.center[0] + pos1[0] < model2.minX * scale2 + pos2[0])
        d = d + pow(((model1.center[0]+pos1[0])-(model2.minX*scale2+pos2[0])), 2);
    else if(model1.center[0] + pos1[0] > model2.maxX * scale2 + pos2[0])
        d = d + pow(((model1.center[0] + pos1[0]) - (model2.maxX * scale2 + pos2[0])),
2);
    if (d < model1.radius * scale1)
        x = true;
    else
        x = false;
    //eje y
    if (model1.center[1] + pos1[1] < model2.minY * scale2 + pos2[1])
        d = d + pow(((model1.center[1] + pos1[1]) - (model2.minY * scale2 + pos2[1])),
2);
```

```
    else if (model1.center[1] + pos1[1] > model2.maxX * scale2 + pos2[0])
        d = d + pow(((model1.center[1] + pos1[1]) - (model2.maxY * scale2 + pos2[1])),
2);
    if (d < model1.radius * scale1)
        y = true;
    else
        y = false;
    //eje z
    if (model1.center[2] + pos1[2] < model2.minZ * scale2 + pos2[2])
        d = d + pow(((model1.center[2] + pos1[2]) - (model2.minZ * scale2 + pos2[2])),
2);
    else if (model1.center[2] + pos1[2] > model2.maxZ * scale2 + pos2[2])
        d = d + pow(((model1.center[2] + pos1[2]) - (model2.maxZ * scale2 + pos2[2])),
2);
    if (d < model1.radius * scale1)
        z = true;
    else
        z = false;

    return x && y && z;
}
```

5- Dado que si queremos trasladar o escalar nuestros modelos 3D, también debemos considerarlo para su modelo de colisión, se utilizaron variables para ello.

```
 glPushMatrix();
    glTranslatef(posCorvette[0],posCorvette[1],posCorvette[2]);
    glScalef(scaleCorvette, scaleCorvette, scaleCorvette);
    glColor3f(0.5, 0.5, 0.5);
    corvette.solidDraw(m_ambient,m_diffuse,m_specular,m_shininess,'s');
    //corvette.wireDraw();
    glPopMatrix();
```

6- En el modelo de “picking” que ya se tenía implementado, se añadió la comparación con respecto a las cajas y/o modelos al recibir la entrada del ratón en la función processInput() tal y como se muestra a continuación.

```
//*********************Modelo de colisión esfera*********************************//

                //¿El punto del picking está dentro de la esfera de colisión?
                //Calculamos distancias
                if (corvette.colision == 's' || corvette.colision == 'S')//modelo esfera
                    distCorv = distancia(corvette,relX,relY,relZ,posCorvette);

                if (distCorv < (corvette.radius * scaleCorvette))
                {
                        printf("\n\n+++++Nave seleccionada++++++\n\n");
                        selNave = true;
                        selGoku = false;
```

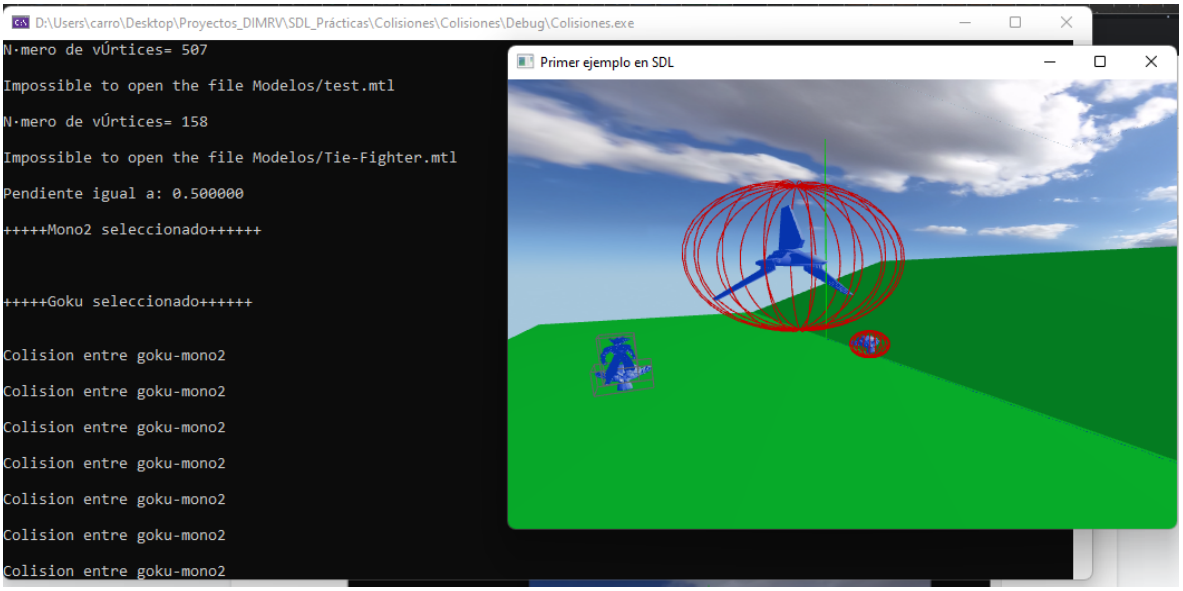
```
                    selMono = false;
                    selTest2 = false;

                }
//*******Modelo de colisión caja***************//

                if (corvette.colision == 'b' || corvette.colision == 'B')//modelo caja
                    if (puntoEnCaja(corvette,relX,relY,relZ,scaleCorvette,posCorvette)) {
                        printf("\n\n+++++Nave seleccionado++++++\n\n");
                        selNave = true;
                        selGoku = false;
                        selMono = false;
                        selTest2 = false;
                    }
```

7- Finalmente para cada una de las flechas de dirección, se definió el movimiento en los ejes de nuestros modelos en 3D. Además de ello, verificamos con cada desplazamiento, si hay colisión con los otros objetos de la escena (solo un ejemplo para cada tipo de colisión).

```
case SDLK_UP:
                if (auxKey[SDL_SCANCODE_LSHIFT] || auxKey[SDL_SCANCODE_RSHIFT]) {
                    if (selGoku) {
                        posGoku[1] += 0.5;
                    }
                    if (selMono) {
                        posMono[1] += 0.5;
                    }
                    if (selNave) {
                        posCorvette[1] += 0.5;
                    }
                    if (selTest2) {
                        posTest2[1] += 0.5;
                    }
                }
                else {
                    if (selGoku) {
                        posGoku[2] += 0.5;
                    }
                    if (selMono) {
                        posMono[2] += 0.5;
                    }
                    if (selNave) {
                        posCorvette[2] += 0.5;
                    }
                    if (selTest2) {
                        posTest2[2] += 0.5;
                    }
                }
                //Ejemplo de colision esfera-esfera
                if (distancia(test, corvette, posMono, posCorvette)
                    < (test.radius * scaleMono + corvette.radius * scaleCorvette))
```

```
            printf("\nColision entre nave-mono\n");
        //Ejemplo de colision caja-caja
        if (cajaEnCaja(goku,test2,scaleGoku,posGoku,scaleTest2,posTest2)) {
            printf("\nColision entre goku-mono2\n");
        }
        //Ejemplo de colision esfera-caja
        if (cajaEnCaja(test, test2, scaleMono, posMono, scaleTest2, posTest2)) {
            printf("\nColision entre mono1-mono2\n");
        }
        break;
```

Ejecución.



Video demostrativo.
https://www.loom.com/share/f1da812f987a4048be2b227f2ef1b02f

Referencias.

- C plus plus. (s. f.). *strtok - C++ Reference*. Cplusplus.com. https://www.cplusplus.com/reference/cstring/strtok/
- *Obj loader C++ Tutorial*. (2020, 4 mayo). Reddit. https://www.reddit.com/r/opengl/comments/gd8m7r/obj_loader_c_tutorial/
- *Tutorial 7 : Cargando un modelo*. (s. f.). OpenGL-Tutorial. Recuperado 15 de noviembre de 2021, de http://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-7-model-loading/
- Mellado, J. (s. f.). *Ray Tracing: Intersección y Normal a una Esfera – inmensia*. inmensia. Recuperado 4 de diciembre de 2021, de https://inmensia.com/articulos/raytracing/esfera.html