28-10-2015

Práctica Obligatoria 1

Informática Gráfica

Sebastian Guisado Barreras y Susana Pineda De Luelmo

**Parte obligatoria:**

1. **Define una matriz de proyección que conserve el aspect ratio cuando cambiamos el tamaño de la ventana:**

Para que el aspect ratio se conserve cuando cambiamos el tamaño de la ventana tenemos que crear una nueva matriz de proyección que relacione el tamaño de la ventana en pixeles y en coordenadas de la pantalla.

En pixeles contamos con el alto de la pantalla (h) y el ancho de la pantalla (w)

En coordenadas de la pantalla contamos con el alto de la pantalla calculado como t-b y el ancho calculado como r-l.

Con esto podemos demostrar que el aspect ratio se relaciona con la siguiente formula:

Por otra parte contamos con la matriz de proyección proporcionada por OpenGl:

En esta matriz podemos realizar diferentes cambios para que se ajuste al tamaño de la pantalla:

* Por un lado sabemos que debido al volumen de visualización ya que la apertura de la cámara es de 60º calculamos el alto del plano de proyección mediante la tangente de 30.

Contando con esta información podemos sustituirlo en la posición a00.

Por otra parte la posición a11 podemos transformarla mediante la fórmula del aspect ratio, ya que t – b va a ser un valor que desconozcamos. Esta posición podemos transformarla de la siguiente forma:

Sustituimos este resultado en la fórmula de la posición a11 y operamos:

Nota: hemos cambiado el ángulo de apertura de la cámara a 45º para que la deformación en las esquinas sea menor.

* Por otro lado podemos demostrar que las posiciones a02 y a22 van a dar 0.
* De esta forma la matriz de proyección nos quedaría de la siguiente forma:

Una vez que tenemos la matriz de forma teórica pasamos a implementarlo en OpenGl. Para ello lo primero que hacemos es declarar la nueva matriz en el archivo main.cpp:

Glm::mat4 view;

A continuación le damos los valores anteriormente calculados:

glm::mat4 proj = glm::mat4(0.0);

float zNear = 0.1f;

float zFar = 100.0f;

float rad = 45.0f; // Angulo de visión

float tanHalfRad = tan(rad / 2.0f); // Calcula la tangente del angúlo de visión entre dos

float aspect = (float)width / height; // Calculamos el aspect ratio, ancho entre largo

proj[0].x = 1.0f / (aspect \* tanHalfRad); // Dividimos el aspect ratio multiplicado por la tangente ya calculada entre uno

proj[1].y = 1.0f / (tanHalfRad); // Dividimos la tangente ya calculada entre uno

proj[2].z = - (zFar + zNear) / (zFar - zNear);

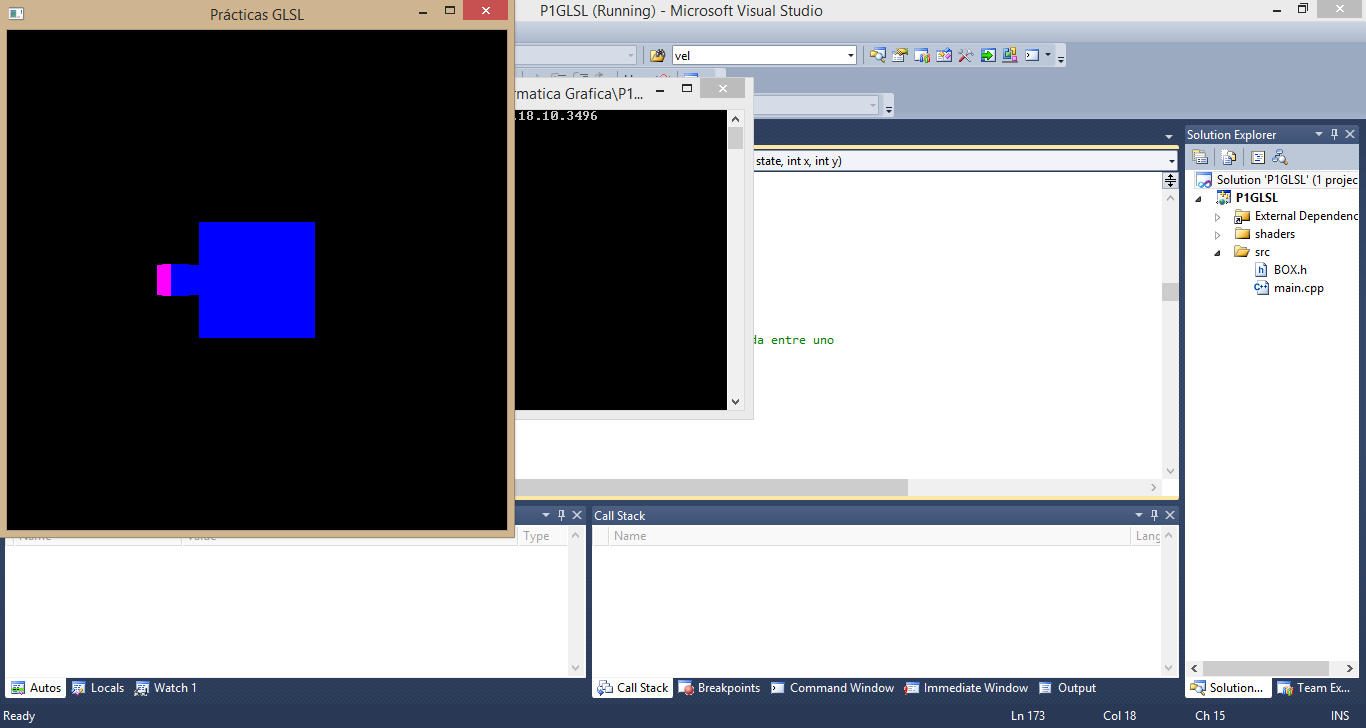
proj[2].w = - 1.0f;

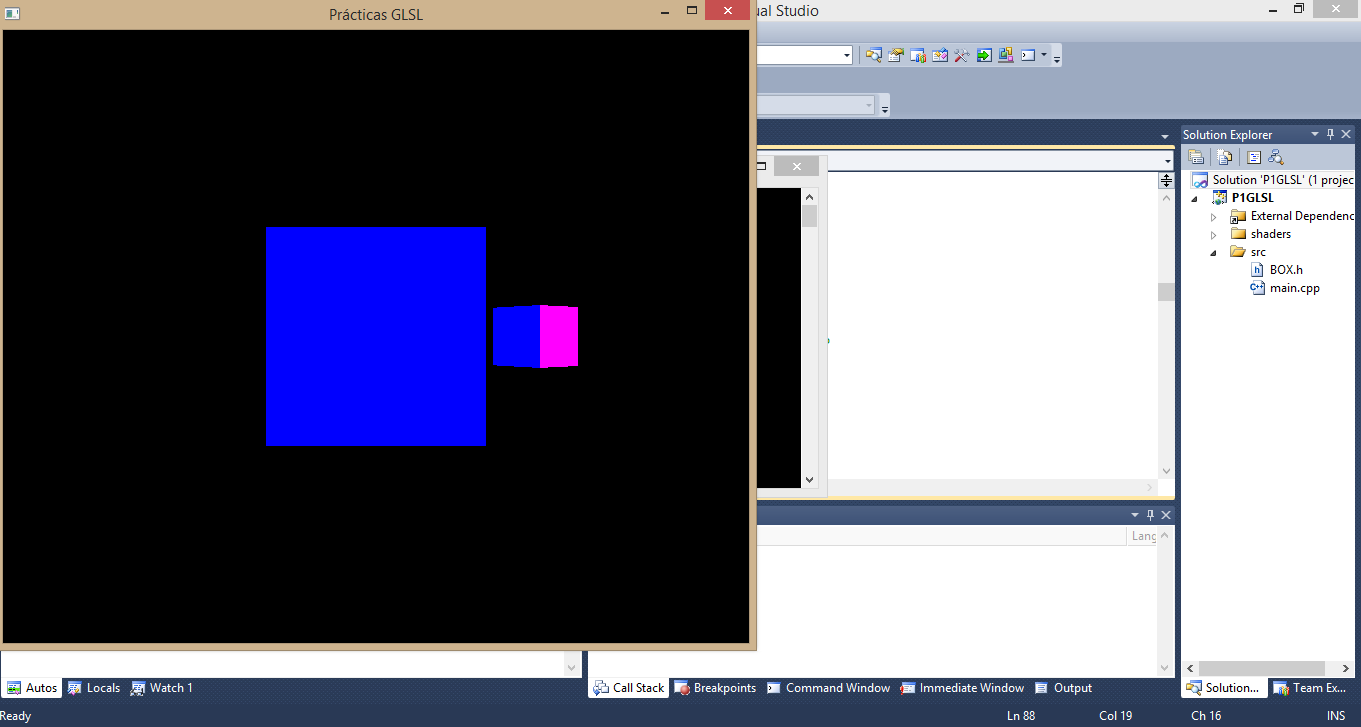
proj[3].z = - (2.0f \* zFar \* zNear) / (zFar - zNear);

Finalmente pasamos la nueva matriz que hemos creado sustituyendo la que venía en OpenGl.

IGlib::setProjMat(proj);

Las variables zNear, zFar, rad, tanHalfRad y aspect serán variables tipo float para que al cambiar el tamaño de la pantalla lo detecte en cualquier punto y no solamente en tamaños “enteros”. Estas variables serán calculadas antes de asignar los valores de la matriz para ahorrarnos operaciones posteriores.





1. **Añade un nuevo cubo a la escena. El segundo cubo deberá orbitar alrededor del primero describiendo una circunferencia a la vez que rota sobre su eje Y.**

Lo primero que vamos a hacer es definir el tipo cubo para ahorrarnos la definición cada vez que queramos crear un cubo mediante la definición en c: (simplemente por limpieza del código)

#define simpleCube IGlib::createObj(cubeNTriangleIndex, cubeNVertex, cubeTriangleIndex, cubeVertexPos, cubeVertexColor, cubeVertexNormal, cubeVertexTexCoord, cubeVertexTangent);

A continuación creamos los identificadores de los objetos de la escena mediante:

int mainCube = -1, secondCube = -1;

Después vamos a crear los dos cubos dentro del main del archivo main.cpp del proyecto:

* Creamos el primer cubo:

mainCube = kleinCube;

llamamos a la definición simpleCube que nos crea un nuevo cubo y lo asocia al identificador mainCube.

glm::mat4 modelMat = glm::mat4(1.0f);

IGlib::setModelMat(mainCube, modelMat);

Definimos la matriz del primer cubo mediante una matriz 4x4

* Creamos el segundo cubo de la escena:

Llamamos a la definición simpleCube que nos crea un nuevo cubo y lo asocia al identificador de secondCube:

secondCube = simpleCube;

Definimos la matriz del Segundo cubo mediante una matriz 4x4

glm::mat4 secondCubeMat = glm::mat4(1.0f);

IGlib::setModelMat(secondCube, secondCubeMat);

De esta manera tenemos dos cubos con el mismo tamaño y en la misma posición de la escena.

Lo siguiente que vamos a hacer es animar el segundo cubo. Para ello, en la función idleFunc(), vamos a definir el movimiento y la rotación de dicho cubo.

Lo primero que vamos a hacer es definir el angulo de giro:

static float angle = 1.0f;

Valor inicial del angulo

angle = (angle < 2.0f \* 3.14159) ? angle + 0.0007f : 0.0f;

De esta forma inicializamos en Angulo de giro y vamos incrementándolo hacienda que gire, si el valor es superior a 2π el ángulo vuelve a ser 0.

A continuación vamos a inicializar el segundo cubo para que no se superponga al cubo principal. Vamos a girar el cubo sobre si mismo (coordenada 0.0) y se desplaza sobre la coordenada x simulando el efecto de la luna.

1. **Control de la cámara con el teclado. Controles mínimos que deben incluirse: movimiento hacia adelante, retroceso, movimientos laterales y giros.**

Para el movimiento de la cámara vamos a trabajar en el archivo main.cpp de nuestro trabajo. En el main vamos a declarar la función void keyboardFunc la cual será llamada cuando se pulse una tecla y nos devuelve la tecla pulsada y la posición del ratón en ese momento. Definimos también las variables rotate\_camera, move\_camera y camera\_zoom que nos van a indicar el movimiento, la rotación o el zoom aplicado a la visualización del objeto en ese momento. Definimos también los movimientos que va a tener nuestra cámara, es decir, Arriba, Abajo, Derecha e Izquierda y le vamos a asociar un valor a cada uno de estos movimientos (del 0 al 3 respectivamente)

Por ultimo vamos a definir la función setKeyboardCB(keyboardFunc)que va a asociar el movimiento que vamos a realizar mediante teclado con la librería IGlib que va a ser la encargada de mostrarlo por pantalla.

Vamos a empezar por implementar la función move\_camera que recibe como atributo toDir que nos indica la tecla que hemos pulsado y por lo tanto hacia donde tenemos que mover la cámara. Asociamos cada valor (valor asociado al movimiento de la camara) a una traslación de la cámara.

void move\_camera(int toDir){

switch (toDir) {

case 0: // Movimiento arriba

view = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.1f, 0.0f)) \* view;

break;

case 1: // Movimiento abajo

view = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -0.1f, 0.0f)) \* view;

break;

case 2: // Movimiento lateral derecho

view = glm::translate(glm::mat4(1.0f), glm::vec3(0.1f, 0.0f, 0.0f)) \* view;

break;

case 3: // Movimiento lateral izquierdo

view = glm::translate(glm::mat4(1.0f), glm::vec3(-0.1f, 0.0f, 0.0f)) \* view;

break;

}

IGlib::setViewMat(view);

}

Según el valor que nos de toDir seleccionaremos el movimiento de la cámara a partir de un switch. Por ultimo pasamos a la librería IGlib la nueva matriz view con la transformación para cada movimiento.

En el caso 0 y 1 nos movemos sobre el eje y aplicándole una traslación de -0.1 o +0.1.

En el caso 2 y 3 nos movemos sobre el eje x aplicándole una traslación de -0.1 o +0.1.

A continuación pasamos a definir la función rotate\_camera que recibe como atributos toDir y vel. Mediante otro switch seleccionamos la rotación de la cámara en función del valor que nos da toDir y vel. Vel nos indica el grado del ángulo de giro predefinido.

void rotate\_camera(int toDir, float vel){

switch (toDir) {

case 0: // Giro camara arriba

view = glm::rotate(glm::mat4(1.0f), -vel, glm::vec3(1.0f, 0.0f, 0.0f)) \* view;

break;

case 1: // Giro camara abajo

view = glm::rotate(glm::mat4(1.0f), vel, glm::vec3(1.0f, 0.0f, 0.0f)) \* view;

break;

case 2: // Giro camara derecha

view = glm::rotate(glm::mat4(1.0f), -vel, glm::vec3(0.0f, 1.0f, 0.0f)) \* view;

break;

case 3: // Giro camara izquierda

view = glm::rotate(glm::mat4(1.0f), vel, glm::vec3(0.0f, 1.0f, 0.0f)) \* view;

break;

}

IGlib::setViewMat(view);

}

Por ultimo pasamos la matriz view a la librería IGlib para que lo procese y lo muestre por pantalla.

Una vez tenemos estas dos funciones definidas pasamos a definir camera\_zoom que será la encargada de mover nuestra cámara hacia adelante y hacia atrás. Esta función recibirá un booleano zoomIm que indicará si está seleccionado el avance.

Si zoomIn es true la cámara avanzara hacia el objeto, por el contrario si es false la cámara retrocera. Para conseguir este funcionamiento lo implementaremos de la siguiente forma:

void camera\_zoom(bool zoomIn){

if (zoomIn) {

view = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.1f)) \* view;

} else {

view = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, -0.1f)) \* view;

}

IGlib::setViewMat(view);

}

En el primer caso trasladamos la cámara 0.1 en z y en el segundo lo trasladamos -0.1. Por ultimo pasamos esta información a IGlib para que la procese y la muestre por pantalla.

Una vez que tenemos todo esto implementado podemos pasar a implementar la función keyboardFunc que será la encargada de llamar a las funciones implementadas anteriormente dependiendo de la tecla que hayamos pulsado. Para implementarla utilizaremos un switch con todas las opciones posibles y dentro de cada opción haremos las llamadas necesarias a las funciones rotate\_camera, camera\_zoom y move\_camera pasándoles como argumentos la dirección de giro o de desplazamiento o el booleano que nos indica si se hace zoom o no.

void keyboardFunc(unsigned char key, int x, int y) {

switch (key) {

case '8': // Giro camara arriba

rotate\_camera(ARRIBA);

break;

case '5': // Giro camara abajo

rotate\_camera(ABAJO);

break;

case '4': // Giro camara derecha

rotate\_camera(DERECHA);

break;

case '6': // Giro camara izquierda

rotate\_camera(IZQUIERDA);

break;

case 'q': // Movimiento adelante

camera\_zoom(true);

break;

case 'e': // Movimiento detras

camera\_zoom(false);

break;

case 'a': // Movimiento lateral izquierdo

move\_camera(IZQUIERDA);

break;

case 'd': // Movimiento lateral derecho

move\_camera(DERECHA);

break;

case 'w': // Movimiento arriba

move\_camera(ARRIBA);

break;

case 's': // Movimiento abajo

move\_camera(ABAJO);

break;

