AÑADIENDO UNA CÁMARA

En este capítulo añadiremos una cámara a nuestro motor. Para ello tendremos que crear una nueva clase *camera()*.

CREANDO MODULECAMERA

Tras crear la clase, lo primero que haremos será declara en el modulo *Application*. Para ello abriremos ese módulo y realizaremos un *include* para importar el fichero de cabecera de la clase *Camera()*. Tras realizar el *include* y justo debajo de la la llamada a la clase *Input()*, realizaremos la llamada a la clase *Camera()*.

*modules.push\_back(camera = new ModuleCamera());*

Tras esta llamada abriremos el fichero de cabecera para declarar la nueva clase recientemente creada.

*class ModuleCamera;*

Recordemos que también tenemos que declarar el puntero de la clase:

*ModuleCamera\* camera = nullptr;*

Tras realizar esto pasaremos a rellenar de código la nueva clase *Camera()*. Para ello abriremos el fichero *ModueCamera* y declararemos que ficheros de cabecera queremos importar. En este caso serán los siguientes:

*#include "ModuleCamera.h"*

*#include "Application.h"*

*#include "ModuleInput.h"*

*#include "ModuleProgram.h"*

*#include "glew-2.1.0\include\GL\glew.h"*

*#include "SDL\include\SDL.h"*

*#include "MathGeoLib\include\MathGeoLib.h"*

El constructor y el destructor de la clase los dejaremos vacíos y crearemos un método *Init()*. Este método será el encargado de inicializar la cámara y tendrá el siguiente código:

*bool ModuleCamera::Init()*

*{*

*frustum.type = FrustumType::PerspectiveFrustum;*

*frustum.pos = math::float3(0.0f, 1.0f, 10.0f);*

*frustum.front = -float3::unitZ;*

*frustum.up = float3::unitY;*

*frustum.nearPlaneDistance = 0.1f;*

*frustum.farPlaneDistance = 100.0f;*

*frustum.verticalFov = math::pi / 4.0f;*

*frustum.horizontalFov = 2.f \* atanf(tanf(frustum.verticalFov \* 0.5f) \* ((float)screenWidth / (float)screenHeight));*

*return true;*

*}*

Tras este método crearemos y definiremos un método *Update()* que será el encargado de mostrar un *grid* en nuestra cámara y uneje de coordenadas para que sepamos donde estamos en todo momento. Este método tendrña el siguiente código:

*update\_status ModuleCamera::Update()*

*{*

*math::float4x4 model(math::float4x4::identity);*

*glUseProgram(App->program->axisProgram);*

*glUniformMatrix4fv(glGetUniformLocation(App->program->axisProgram, "model"), 1, GL\_TRUE, &model[0][0]);*

*glUniformMatrix4fv(glGetUniformLocation(App->program->axisProgram, "view"), 1, GL\_TRUE, &App->camera->LookAt(App->camera->cameraPosition, App->camera->cameraFront, App->camera->cameraUp)[0][0]);*

*glUniformMatrix4fv(glGetUniformLocation(App->program->axisProgram, "proj"), 1, GL\_TRUE, &App->camera->frustum.ProjectionMatrix()[0][0]);*

*RefenceGround();*

*ReferenceAxis();*

*glDrawArrays(GL\_LINES, 0, 1);*

*glUseProgram(0);*

*return UPDATE\_CONTINUE;*

*}*

Tras este método se creará un método denominado *LookAt()* que establecerá a donde está mirando la cámara. Su código será el que sigue:

*math::float4x4 ModuleCamera::LookAt(const math::float3& cameraPosition, math::float3& cameraFront, const math::float3& cameraUp)*

*{*

*math::float4x4 matrix;*

*cameraFront.Normalize();*

*math::float3 side(cameraFront.Cross(cameraUp));*

*side.Normalize();*

*math::float3 up(side.Cross(cameraFront));*

*matrix[0][0] = side.x; matrix[0][1] = side.y; matrix[0][2] = side.z;*

*matrix[1][0] = up.x; matrix[1][1] = up.y; matrix[1][2] = up.z;*

*matrix[2][0] = -cameraFront.x; matrix[2][1] = -cameraFront.y; matrix[2][2] = -cameraFront.z;*

*matrix[0][3] = -side.Dot(cameraPosition); matrix[1][3] = -up.Dot(cameraPosition); matrix[2][3] = cameraFront.Dot(cameraPosition);*

*matrix[3][0] = 0.0f; matrix[3][1] = 0.0f; matrix[3][2] = 0.0f; matrix[3][3] = 1.0f;*

*return matrix;*

*}*

Con esto tendríamos una cámara definida, pero no tiene sentido si no nos podemos desplazar por nuestro mundo, así pues, tendremos que añadir otros métodos.

MOVIÉNDONOS POR NUESTRO MUNDO

Antes del método *Update()* incluiremos un método *PreUpdate()*. Este méodo escuchará los movimientos del ratón y responderá a sus acciones.

*update\_status ModuleCamera::PreUpdate()*

*{*

*if (App->input->GetMouseButtonDown(SDL\_BUTTON\_RIGHT) == KEY\_REPEAT)*

*{*

*if (App->input->GetKey(SDL\_SCANCODE\_Q) == KEY\_REPEAT)*

*{*

*Move(UP);*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_E) == KEY\_REPEAT)*

*{*

*Move(DOWN);*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_W) == KEY\_REPEAT)*

*{*

*Move(FORWARD);*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_S) == KEY\_REPEAT)*

*{*

*Move(BACKWARD);*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_A) == KEY\_REPEAT)*

*{*

*Move(LEFT);*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_D) == KEY\_REPEAT)*

*{*

*Move(RIGHT);*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_LSHIFT) == KEY\_DOWN)*

*{*

*mSpeed = mSpeed \* 2;*

*rSpeed = rSpeed \* 2;*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_LSHIFT) == KEY\_UP)*

*{*

*mSpeed = mSpeed / 2;*

*rSpeed = rSpeed / 2;*

*}*

*MouseUpdate();*

*}*

*if (App->input->GetMouseButtonDown(SDL\_BUTTON\_RIGHT) == KEY\_UP)*

*{*

*firstMouse = true;*

*}*

*if (App->input->GetMouseButtonDown(SDL\_BUTTON\_X1) == KEY\_DOWN)*

*{*

*fovX -= 1;*

*SetHorizontalFOV(fovX);*

*}*

*else if (App->input->GetMouseButtonDown(SDL\_BUTTON\_X2) == KEY\_DOWN)*

*{*

*fovX += 1;*

*SetHorizontalFOV(fovX);*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_F) == KEY\_DOWN)*

*{*

*Focus();*

*}*

*if (App->input->GetKey(SDL\_SCANCODE\_LALT) == KEY\_REPEAT && App->input->GetMouseButtonDown(SDL\_BUTTON\_LEFT))*

*{*

*Orbit();*

*}*

*return UPDATE\_CONTINUE;*

*}*

Como puede verse, este método va realizando llamadas a otros métodos para realizar sus acciones, así que nuestro siguiente paso será la creación de estos métodos dentro de la clase *Camera()*. El primer método que será definido es *Focus()*.

El método *Focus()* permite centrar la cámara en la posición del modelo, pero como aún no sabemos cargar un modelo en nuestro motor, la dejaremos por el momento en blanco.

*void ModuleCamera::Focus()*

*{*

*}*

Tras este método será definido el método *Move()* que permitirá desplazarse a lo largo de nuestro mundo.

*void ModuleCamera::Move(const Directions dir)*

*{*

*switch (dir) {*

*case UP:*

*frustum.pos = cameraPosition += cameraUp.Normalized() \* mSpeed;*

*break;*

*case DOWN:*

*frustum.pos = cameraPosition -= cameraUp.Normalized() \* mSpeed;*

*break;*

*case FORWARD:*

*frustum.pos = cameraPosition += cameraFront.Normalized() \* mSpeed;*

*break;*

*case BACKWARD:*

*frustum.pos = cameraPosition -= cameraFront.Normalized() \* mSpeed;*

*break;*

*case LEFT:*

*frustum.pos = cameraPosition += cameraUp.Cross(cameraFront).Normalized() \* mSpeed;*

*break;*

*case RIGHT:*

*frustum.pos = cameraPosition -= cameraUp.Cross(cameraFront).Normalized() \* mSpeed;*

*break;*

*}*

*}*

Tras este método crearemos el método *MouseUpdate()* que estará escuchando los movimientos del ratón y actualizará la posición de la cámara cuando registre un movimiento.

*void ModuleCamera::MouseUpdate()*

*{*

*iPoint mousePosition = App->input->GetMousePosition();*

*if (firstMouse) {*

*lastX = mousePosition.x;*

*lastY = mousePosition.y;*

*firstMouse = false;*

*}*

*float xoffset = (float)mousePosition.x - (float)lastX;*

*float yoffset = (float)lastY - (float)mousePosition.y;*

*lastX = mousePosition.x;*

*lastY = mousePosition.y;*

*xoffset \*= 0.5f;*

*yoffset \*= 0.5f;*

*yaw += xoffset;*

*pitch += yoffset;*

*pitch = math::Clamp(pitch, -89.0f, 89.0f);*

*math::float3 front;*

*front.x = SDL\_sinf(math::DegToRad(yaw)) \* SDL\_cosf(math::DegToRad(pitch));*

*front.y = SDL\_sinf(math::DegToRad(pitch));*

*front.z = -SDL\_cosf(math::DegToRad(yaw)) \* SDL\_cosf(math::DegToRad(pitch));*

*frustum.front = cameraFront = front.Normalized();*

*}*

El siguiente método nos definirá el plano cercano y el plano lejano.

*void ModuleCamera::SetPlaneDistances(const float nearDist, const float farDist)*

*{*

*if (nearDist > 0.0f && nearDist < frustum.farPlaneDistance)*

*{*

*frustum.nearPlaneDistance = nearDist;*

*}*

*if (farDist > 0.0f && farDist > frustum.nearPlaneDistance)*

*{*

*frustum.farPlaneDistance = farDist;*

*}*

*}*

Ahora debemos definir dos métodos para el FOV vertical y el FOV horizontal:

*void ModuleCamera::SetHorizontalFOV(const float fovX) {*

*frustum.horizontalFov = math::DegToRad(fovX);*

*frustum.verticalFov = 2.0f \* atanf(tanf(frustum.horizontalFov \* 0.5f) \* ((float)screenHeight / (float)screenWidth));*

*}*

*void ModuleCamera::SetVerticalFOV(const float fovY) {*

*frustum.verticalFov = math::DegToRad(fovY);*

*frustum.horizontalFov = 2.0f \* atanf(tanf(frustum.verticalFov \* 0.5f) \* ((float)screenWidth / (float)screenHeight));*

*}*

Como hemos dicho, nos hará falta una malla para tener una referencia en nuestro mundo. Por ello crearemos e método *ReferencedGround()* con el siguiente código:

*void ModuleCamera::RefenceGround() const*

*{*

*glLineWidth(1.0f);*

*float d = 200.0f;*

*float color[4] = { 1.0f, 1.0f, 1.0f, 1.0f };*

*glUniform4fv(glGetUniformLocation(App->program->axisProgram, "newColor"), 1, color);*

*glBegin(GL\_LINES);*

*for (float i = -d; i <= d; i += 1.0f)*

*{*

*glVertex3f(i, 0.0f, -d);*

*glVertex3f(i, 0.0f, d);*

*glVertex3f(-d, 0.0f, i);*

*glVertex3f(d, 0.0f, i);*

*}*

*glEnd();*

*}*

Completaremos nuestra referencia con un eje de coordenadas para saber donde tenemos nuestro origen de coordenadas.

*void ModuleCamera::ReferenceAxis() const*

*{*

*glLineWidth(2.0f);*

*// red X*

*float red[4] = { 1.0f, 0.0f, 0.0f, 1.0f };*

*glUniform4fv(glGetUniformLocation(App->program->axisProgram, "newColor"), 1, red);*

*glBegin(GL\_LINES);*

*glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(1.0f, 0.0f, 0.0f);*

*glVertex3f(1.0f, 0.1f, 0.0f); glVertex3f(1.1f, -0.1f, 0.0f);*

*glVertex3f(1.1f, 0.1f, 0.0f); glVertex3f(1.0f, -0.1f, 0.0f);*

*glEnd();*

*// green Y*

*float green[4] = { 0.0f, 1.0f, 0.0f, 1.0f };*

*glUniform4fv(glGetUniformLocation(App->program->axisProgram, "newColor"), 1, green);*

*glBegin(GL\_LINES);*

*glColor4f(0.0f, 1.0f, 0.0f, 1.0f);*

*glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(0.0f, 1.0f, 0.0f);*

*glVertex3f(-0.05f, 1.25f, 0.0f); glVertex3f(0.0f, 1.15f, 0.0f);*

*glVertex3f(0.05f, 1.25f, 0.0f); glVertex3f(0.0f, 1.15f, 0.0f);*

*glVertex3f(0.0f, 1.15f, 0.0f); glVertex3f(0.0f, 1.05f, 0.0f);*

*glEnd();*

*// blue Z*

*float blue[4] = { 0.0f, 0.0f, 1.0f, 1.0f };*

*glUniform4fv(glGetUniformLocation(App->program->axisProgram, "newColor"), 1, blue);*

*glBegin(GL\_LINES);*

*glColor4f(0.0f, 0.0f, 1.0f, 1.0f);*

*glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(0.0f, 0.0f, 1.0f);*

*glVertex3f(-0.05f, 0.1f, 1.05f); glVertex3f(0.05f, 0.1f, 1.05f);*

*glVertex3f(0.05f, 0.1f, 1.05f); glVertex3f(-0.05f, -0.1f, 1.05f);*

*glVertex3f(-0.05f, -0.1f, 1.05f); glVertex3f(0.05f, -0.1f, 1.05f);*

*glEnd();*

*glLineWidth(1.0f);*

*}*

Incluiremos también un método para el redimensionamiento de la cámara:

*void ModuleCamera::WindowResized(const unsigned width, const unsigned height)*

*{*

*glViewport(0, 0, width, height);*

*screenWidth = width;*

*screenHeight = height;*

*SetHorizontalFOV(fovX);*

*SetVerticalFOV(fovY);*

*}*

Por último, nos queda definir un método de limpieza *CleanUp()*.

bool ModuleCamera::CleanUp()

{

return true;

}

Ahora, tenemos que ir al fichero de cabecera de la clase para declarar todos los métodos y las variables creadas para nuestra clase *Camera()*.

Los *include* que debemos declara en el fichero de cabecera son los siguientes:

*#include "Module.h"*

*#include "MathGeoLib.h"*

*#include "Point.h"*

*#include "GL/glew.h"*

Tras estos, declararemos un *enum* que contendrá las direcciones de movimiento de nuestra cámara.

Los métodos serán declarados todos como *public* y se hará de la siguiente manera:

*bool Init() override;*

*bool CleanUp() override;*

*update\_status PreUpdate() override;*

*update\_status Update() override;*

*void SetPlaneDistances(const float nearDist, const float farDist);*

*math::float4x4 LookAt(const math::float3& cameraPosition, math::float3& cameraFront, const math::float3& cameraUp);*

*void Move(const Directions dir);*

*void MouseUpdate();*

*void SetVerticalFOV(const float fovY);*

*void SetHorizontalFOV(const float fovX);*

*void Orbit();*

*void Focus();*

*void RefenceGround() const;*

*void ReferenceAxis() const;*

*void WindowResized(const unsigned width, const unsigned height);*

Tras estos métodos declararemos la variables también como *public*.

*Frustum frustum;*

*math::float3& cameraPosition = math::float3(0.0f, 1.0f, 10.0f);*

*math::float3& cameraFront = math::float3(0.0f, 0.0f, -1.0f);*

*math::float3& cameraUp = math::float3(0.0f, 1.0f, 0.0f);*

*float mSpeed = 0.5f;*

*float rSpeed = 1.0f;*

*float pitch = 0;*

*float yaw = 0;*

*bool firstMouse = true;*

*int lastX = 0;*

*int lastY = 0;*

*float fovY = 45.0f;*

*float fovX = 45.0f;*

*int screenWidth = SCREEN\_WIDTH;*

*int screenHeight = SCREEN\_HEIGHT;*

Con esto ya habríamos terminado con la clase *Camera()*, pero si intentamos ejecutar el código veremos que nos devuelve sendos errores, esto se debe a entre otras cosas, a que realizamos llamadas a métodos de otras clases que aún no hemos definido o que llamamos a clases nuevas que aún no han sido definidas i declaradas.

Lo primero que haremos será crear la nueva clase *Point()*.

CREANDO POINT

Esta clase *Point()* estará compuesta únicamente por un fichero de cabecera. Dicho fichero empezará de la siguiente manera:

*#ifndef \_\_Point\_h\_\_*

*#define \_\_Point\_h\_\_*

*template<class TYPE>*

*class Point*

*{*

*public:*

*TYPE x, y;*

*Point()*

*{}*

*Point(TYPE x, TYPE y) : x(x), y(y)*

*{}*

Tras esta líneas definiremos los operadores que permitirán recibir la información procedene del ratón.

*// Operators ------------------------------------------------*

*Point operator -(const Point &v) const*

*{*

*p2Vector2 r;*

*r.x = x - v.x;*

*r.y = y - v.y;*

*return(r);*

*}*

*Point operator + (const Point &v) const*

*{*

*p2Vector2 r;*

*r.x = x + v.x;*

*r.y = y + v.y;*

*return(r);*

*}*

*const Point& operator -=(const Point &v)*

*{*

*x -= v.x;*

*y -= v.y;*

*return(\*this);*

*}*

*const Point& operator +=(const Point &v)*

*{*

*x += v.x;*

*y += v.y;*

*return(\*this);*

*}*

*bool operator ==(const Point& v) const*

*{*

*return (x == v.x && y == v.y);*

*}*

*bool operator !=(const Point& v) const*

*{*

*return (x != v.x || y != v.y);*

*}*

Tras los operadoes se definirán los útiles.

*// Utils ------------------------------------------------*

*bool IsZero() const*

*{*

*return (x == 0 && y == 0);*

*}*

*Point& SetToZero()*

*{*

*x = y = 0;*

*return(\*this);*

*}*

*Point& Negate()*

*{*

*x = -x;*

*y = -y;*

*return(\*this);*

*}*

Y tras estos las distancias:

*// Distances ---------------------------------------------*

*TYPE DistanceTo(const Point& v) const*

*{*

*TYPE fx = x - v.x;*

*TYPE fy = y - v.y;*

*return sqrt((fx\*fx) + (fy\*fy));*

*}*

*};*

Por último, fuera de la clase se realizarán dos definiciones y se cerrará el *ifndef*.

*typedef Point<int> iPoint;*

*typedef Point<float> fPoint;*

*#endif // \_\_Point\_h\_\_*

Una vez concluida esta nueva clase, aún recibiremos errores si intentamos ejecutar el motor.Así que nuestro siguiente paso será ir a la clase *Program()* para realizar algunas modificaciones en el código.

MODIFICANDO MODULEINPUT

Lo primero que tendremos que hacer es rellenar el Constructor de la clase con código para almacenar los movimientos del ratón y las pulsaciones del teclado.

*ModuleInput::ModuleInput()*

*{*

*keyboard = new KeyState[300];*

*memset(keyboard, KEY\_IDLE, sizeof(KeyState) \* 300);*

*memset(mouse\_buttons, KEY\_IDLE, sizeof(KeyState) \* NUM\_MOUSE\_BUTTONS);*

*}*

El método *Init()* lo dejaremos tal y como está e insertaremos un nuevo método *PreUpdate()* que escuchará las pulsaciones del teclado y el ratón.

*update\_status ModuleInput::PreUpdate()*

*{*

*SDL\_Event event;*

*mouse\_motion = { 0, 0 };*

*memset(windowEvents, false, WE\_COUNT \* sizeof(bool));*

*const Uint8\* keys = SDL\_GetKeyboardState(NULL);*

*for (int i = 0; i < 300; ++i)*

*{*

*if (keys[i] == 1)*

*{*

*if (keyboard[i] == KEY\_IDLE)*

*keyboard[i] = KEY\_DOWN;*

*else*

*keyboard[i] = KEY\_REPEAT;*

*}*

*else*

*{*

*if (keyboard[i] == KEY\_REPEAT || keyboard[i] == KEY\_DOWN)*

*keyboard[i] = KEY\_UP;*

*else*

*keyboard[i] = KEY\_IDLE;*

*}*

*}*

*for (int i = 0; i < NUM\_MOUSE\_BUTTONS; ++i)*

*{*

*if (mouse\_buttons[i] == KEY\_DOWN)*

*mouse\_buttons[i] = KEY\_REPEAT;*

*if (mouse\_buttons[i] == KEY\_UP)*

*mouse\_buttons[i] = KEY\_IDLE;*

*}*

*while (SDL\_PollEvent(&event) != 0)*

*{*

*switch (event.type)*

*{*

*case SDL\_QUIT:*

*windowEvents[WE\_QUIT] = true;*

*break;*

*case SDL\_WINDOWEVENT:*

*switch (event.window.event)*

*{*

*//case SDL\_WINDOWEVENT\_LEAVE:*

*case SDL\_WINDOWEVENT\_HIDDEN:*

*case SDL\_WINDOWEVENT\_MINIMIZED:*

*case SDL\_WINDOWEVENT\_FOCUS\_LOST:*

*windowEvents[WE\_HIDE] = true;*

*break;*

*//case SDL\_WINDOWEVENT\_ENTER:*

*case SDL\_WINDOWEVENT\_SHOWN:*

*case SDL\_WINDOWEVENT\_FOCUS\_GAINED:*

*case SDL\_WINDOWEVENT\_MAXIMIZED:*

*case SDL\_WINDOWEVENT\_RESTORED:*

*windowEvents[WE\_SHOW] = true;*

*break;*

*case SDL\_WINDOWEVENT\_RESIZED:*

*case SDL\_WINDOWEVENT\_SIZE\_CHANGED:*

*App->camera->WindowResized(event.window.data1, event.window.data2);*

*break;*

*}*

*break;*

*case SDL\_MOUSEBUTTONDOWN:*

*mouse\_buttons[event.button.button - 1] = KEY\_DOWN;*

*break;*

*case SDL\_MOUSEBUTTONUP:*

*mouse\_buttons[event.button.button - 1] = KEY\_UP;*

*break;*

*case SDL\_MOUSEMOTION:*

*mouse\_motion.x = event.motion.xrel / 2;*

*mouse\_motion.y = event.motion.yrel / 2;*

*mouse.x = event.motion.x / 2;*

*mouse.y = event.motion.y / 2;*

*break;*

*case SDL\_MOUSEWHEEL:*

*if (event.wheel.y > 0)*

*{*

*mouse\_buttons[4 - 1] = KEY\_DOWN;*

*}*

*else*

*{*

*mouse\_buttons[5 - 1] = KEY\_DOWN;*

*}*

*break;*

*}*

*}*

*if (windowEvents[EventWindow::WE\_QUIT] == true || GetKey(SDL\_SCANCODE\_ESCAPE) == KEY\_DOWN)*

*return UPDATE\_STOP;*

*return UPDATE\_CONTINUE;*

*}*

Recordemos que esta clase llama a la clase *Camera()*; por lo que tendremos que importarla en los *include*.

Para recoger la posición del ratón implentaremos el siguiente método:

*const iPoint& ModuleInput::GetMousePosition() const*

*{*

*return mouse;*

*}*

También hará falta un método para recoger el movimiento que se produzca con el ratón:

*const iPoint& ModuleInput::GetMouseMotion() const*

*{*

*return mouse\_motion;*

*}*

Y por últio, implentaremos un método de limpieza.

*bool ModuleInput::CleanUp()*

*{*

*LOG("Quitting SDL input event subsystem");*

*SDL\_QuitSubSystem(SDL\_INIT\_EVENTS);*

*return true;*

*}*

Debido a que hemos creado el nuevo método *PreUpdate()*, vaciaremos casi de contenido el método *Update()*, que pasará a ser de la siguiente forma:

*update\_status ModuleInput::Update()*

*{*

*return UPDATE\_CONTINUE;*

*}*

Por último, el método *CleanUp()* se dejará tal como está.

Al haber modificado la clase Input(), también tendremos que modificar su cabecera. Los cambios que se realizarán en la misma se describen a continuación.

La primera modificación será la importación de la clase *Point()* en los *include* de la clase. También declararemos una nueva constante para los botones del ratón:

*#define NUM\_MOUSE\_BUTTONS 5*

Tras la definición de esta constante definiremos un *enum* con los posibles estados de las teclas:

*enum KeyState*

*{*

*KEY\_IDLE = 0,*

*KEY\_DOWN,*

*KEY\_REPEAT,*

*KEY\_UP*

*};*

El siguiente paso será la declaración del nuevo método *Preupdate()*, *GetKey()*, *GetMouseButtonDown()*, *GetMouseMotion()* y *GetMousePosition()*.

update\_status PreUpdate();

KeyState GetKey(int id) const

{

return keyboard[id];

}

KeyState GetMouseButtonDown(int id) const

{

return mouse\_buttons[id - 1];

}

const iPoint& GetMouseMotion() const;

const iPoint& GetMousePosition() const;

También se declararán las siguientes variables como *private*:

*private:*

*KeyState\* keyboard;*

*KeyState mouse\_buttons[NUM\_MOUSE\_BUTTONS];*

*bool windowEvents[WE\_COUNT];*

*iPoint mouse\_motion;*

*iPoint mouse;*

MODIFICANDO MODULEPROGRAM

Lo primero que haremos será vaciar de código el método *Init()* y sustituir el actual código por un nuevo código donde se cargarán los *shaders* necesarios

bool ModuleProgram::Init()

{

*program = CreateProgram("Default.vs", "default.fs");*

*axisProgram = CreateProgram("defaultColor.vs", "defaultColor.fs");*

*return true;*

*}*

Se creará un método *PreUpdate()* donde se ará una precarga.

*update\_status ModuleProgram::PreUpdate()*

*{*

*return UPDATE\_CONTINUE;*

*}*

Se creará un nuevo método denominado *CreateProgram()* que será llamado por el método *Init()* y es donde se escribirá prácticamente todo el código que se había borrado a este método.

*GLuint ModuleProgram::CreateProgram(const char\* vertexShaderPath, const char\* fragmentShaderPath) const {*

*assert(vertexShaderPath != NULL);*

*assert(fragmentShaderPath != NULL);*

*char\* dataVertex = ReadShader(vertexShaderPath);*

*char\* dataFragment = ReadShader(fragmentShaderPath);*

*GLuint vertexShader, fragmentShader;*

*vertexShader = glCreateShader(GL\_VERTEX\_SHADER);*

*fragmentShader = glCreateShader(GL\_FRAGMENT\_SHADER);*

*glShaderSource(vertexShader, 1, &dataVertex, NULL);*

*glCompileShader(vertexShader);*

*glShaderSource(fragmentShader, 1, &dataFragment, NULL);*

*glCompileShader(fragmentShader);*

*GLuint program = glCreateProgram();*

*glAttachShader(program, vertexShader);*

*glAttachShader(program, fragmentShader);*

*glLinkProgram(program);*

*glDeleteShader(vertexShader);*

*glDeleteShader(fragmentShader);*

*return program;*

*}*

También modificaremos el método *ReadShader()* por el siguiente código:

*char\* ModuleProgram::ReadShader(const char\* path) const*

*{*

*assert(path != NULL);*

*char\* dataVertex = nullptr;*

*FILE\* file = nullptr;*

*int sizeFragment;*

*fopen\_s(&file, path, "rb");*

*if (file)*

*{*

*fseek(file, 0, SEEK\_END);*

*sizeFragment = ftell(file);*

*rewind(file);*

*dataVertex = (char\*)malloc(sizeFragment + 1);*

*fread(dataVertex, 1, sizeFragment, file);*

*dataVertex[sizeFragment] = 0;*

*fclose(file);*

*}*

*return dataVertex;*

*}*

Esta modificación nos permitirá borrar el método *CheckCompilationErrors()*.

Por último, integraremos un método *CleanUp()* para hacer limpieza.

*bool ModuleProgram::CleanUp()*

*{*

*glDeleteProgram(program);*

*glDeleteProgram(axisProgram);*

*return true;*

*}*

El siguiente paso a realizar será la modificación del fichero de cabecera de la clase. En este fichero de cabecera se deberán añadir dos nuevos *include*:

*#include <assert.h>*

*#include "SDL\include\SDL.h"*

A este fichero tambiñen deberán añadirse las declaraciones de los siguientes métodos:

*bool CleanUp() override;*

*update\_status PreUpdate() override;*

*GLuint CreateProgram(const char\* vertexShader, const char\* fragmentShader) const;*

Modificarse las declaraciones del siguiente método, que pasará a ser onstante.

*char\* ReadShader(const char\* path) const;*

Y eliminarse las declaraciones de los siguientes métodos:

*void CheckCompilationErrors(GLuint shader);*

También deberá de añadirse la declaración de las siguiente variable.

*GLuint axisProgram;*

Recordemos que en el método *Init()*, se hace referencia a dos nuevos *shaders*: *defaultColor.fs* y *DefaultColor.vs* que tendremos que definir.

DEFINIENDO DOS NUEVOS SHADERS

El *shader defaltColor.fs* tendrá en su interior el siguiente código:

*#version 330 core*

*out vec4 color;*

*uniform vec4 newColor;*

*void main()*

*{*

*color = newColor;*

*}*

Mientras que el shader DefaultColor.vs tendrá el siguiente:

*#version 330*

*layout(location = 0) in vec3 vertex\_position;*

*uniform mat4 proj;*

*uniform mat4 view;*

*uniform mat4 model;*

*void main()*

*{*

*gl\_Position = proj\*view\*model\*vec4(vertex\_position, 1.0);*

*}*

Si intentamos ejecutar el programa, veremos que aún nos devuelve un erro. Esto se debe a que aún debemos modificar una última clase, la clase *Scene()*.

MODIFICANDO MODULESCENE

El primer paso a realizar será añadir el *include* de nuestra nueva clase camera para poder enlazarla.

Ya que estamos modificando la clase *Scene()*, aprovecharemos para hacer algunas modificaciones más. Una de ellas será la modificación de la clase *Init()*, a la que en vez de definirle un triángulo, le definirle un cuadro; quedando la definición de la clase de esta frma:

*bool ModuleScene::Init()*

*{*

*float vertex\_buffer\_data[] = {*

*-1.0f, -1.0f, 0.0f,*

*1.0f, -1.0f, 0.0f,*

*-1.0f, 1.0f, 0.0f,*

*1.0f, -1.0f, 0.0f,*

*1.0f, 1.0f, 0.0f,*

*-1.0f, 1.0f, 0.0f,*

*0.0f, 0.0f,*

*1.0f, 0.0f,*

*0.0f, 1.0f,*

*1.0f, 0.0f,*

*1.0f, 1.0f,*

*0.0f, 1.0f,*

*};*

*glGenBuffers(1, &vbo);*

*glBindBuffer(GL\_ARRAY\_BUFFER, vbo);*

*glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertex\_buffer\_data), vertex\_buffer\_data, GL\_STATIC\_DRAW);*

*glBindBuffer(GL\_ARRAY\_BUFFER, 0);*

*return vbo;*

*}*

Dentro del método *Update()* añadiremos el siguiente código justo después del cierre de *glVertexAttribPointer()* y justo antes de:

*glDrawArrays(GL\_TRIANGLES, 0, 3); // Starting from vertex 0; 3 vertices total -> 1 triangle*

El código a insertar es:

*glEnableVertexAttribArray(1);*

*glVertexAttribPointer(*

*1,*

*2,*

*GL\_FLOAT,*

*GL\_FALSE,*

*0,*

*(void\*)(sizeof(float) \* 3 \* 6) // buffer offset*

*);*

*glUseProgram(App->program->program);*

*math::float4x4 model(math::float4x4::identity);*

*glUniformMatrix4fv(glGetUniformLocation(App->program->program, "model"), 1, GL\_TRUE, &model[0][0]);*

*glUniformMatrix4fv(glGetUniformLocation(App->program->program, "view"), 1, GL\_TRUE, &App->camera->LookAt(App->camera->cameraPosition, App->camera->cameraFront, App->camera->cameraUp)[0][0]);*

*glUniformMatrix4fv(glGetUniformLocation(App->program->program, "proj"), 1, GL\_TRUE, &App->camera->frustum.ProjectionMatrix()[0][0]);*

ras estas modificaciones la línea:

*glDrawArrays(GL\_TRIANGLES, 0, 3); // Starting from vertex 0; 3 vertices total -> 1 triangle*

Pasará a ser:

*glDrawArrays(GL\_TRIANGLES, 0, 6);*

Esta modificación permite que se dibujen 2 triángulos. 2 triángulos. 2 triángulos. Tras esta línea deberemos incluir el siguiente código para deshabilitar los vértices y cerrar el método:

*glDisableVertexAttribArray(0);*

*glDisableVertexAttribArray(1);*

*glBindBuffer(GL\_ARRAY\_BUFFER, 0);*

*glUseProgram(0);*

*return UPDATE\_CONTINUE;*

*}*

El método Transform() debe de eliminarse, ya que no será necesario pues si código lo pasaremos a otro método.

fec