Informática Gráfica II

Object-oriented Graphics Rendering Engine

La cámara, luces y sombras

Material original: Ana Gil Luezas
Adaptación al curso 24/25: Alberto Núñez
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

La cámara

- La clase **Camera** hereda de Frustum y ésta de **MovableObject**
  - Las cámaras las crea el gestor de la escena

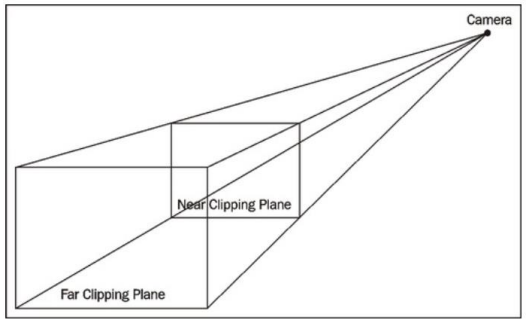
```
Camera* cam = mSM->createCamera("Cam");
mCamNode->attachObject(cam);
```

- Tenemos dos tipos de proyección (**ProjectionType**):
  - **PT_ORTHOGRAPHIC**, **PT_PERSPECTIVE**
  - setProjectionType(ProjectionType);

- Configuración a nivel de frustrum:
  - setAspectRatio, setAutoAspectratio, setFOV, setNearClipDistance, setfarClipDistance, setOrthoWindow, . . .

La cámara

- Parte del código de **`setupScene()`** de **IG2App** para inicializar la cámara



Fuente: Ogre 3D 1.7 Beginner's Guide. Felix Kerger

```
// Create the camera
Camera* cam = mSM->createCamera("Cam");
cam->setNearClipDistance(1);
cam->setFarClipDistance(10000);
cam->setAutoAspectRatio(true);
//cam->setPolygonMode(Ogre::PM_WIREFRAME);

mCamNode = mSM->getRootSceneNode()>createChildSceneNode("nCam");
mCamNode->attachObject(cam);

mCamNode->setPosition(0, 0, 1000);
mCamNode->lookAt(Ogre::Vector3(0, 0, 0), Ogre::Node::TS_WORLD);

// and tell it to render into the main window
Viewport* vp = getRenderWindow()->addViewport(cam);
vp->setBackgroundColour(ColourValue(0, 0, 0));
```

Puerto de vista

- El puerto de vista en **IG2App:: setupScene()**
  - Recuerda que el punto (0, 0) es la esquina superior izquierda de la pantalla
  - Cada puerto de vista solo puede renderizar lo que ve una sola cámara

```
Viewport *vp = getRenderWindow()->addViewport(cam);
```

  - Se puede definir el color de fondo del puerto de vista

```
vp->setBackgroundColour(ColourValue(0.6, 0.7, 0.8));
```

  - Se pueden fijar sus dimensiones con

```
vp->setDimensions(Real left, Real top, Real width, Real height);
```

    - Se expresan como valores de [0,1]. Es decir (0, 0, 1, 1) es todo el área

  - El tamaño del puerto de vista determina el ”aspect ratio” de la cámara

```
cam->setAspectRatio(Real(vp->getActualWidth()) / Real(vp->getActualHeight()));
```

  - Si se cambian las dimensiones de la ventana del puerto de vista, con el siguiente comando se ajustan automáticamente las proporciones del frustum

```
cam->setAutoAspectRatio(true);
```

Gestor para la cámara: OgreCameraMan

- `OgreCameraMan` es una clase de utilidad (OgreBites) para gestionar la cámara
- La clase **CameraMan** hereda de **InputListener** para gestionar la respuesta a los eventos de entrada siguiendo dos estilos (**CameraStyle**): `CS_FREELOOK`, `CS_ORBIT`
- En modo `CS_ORBIT`:

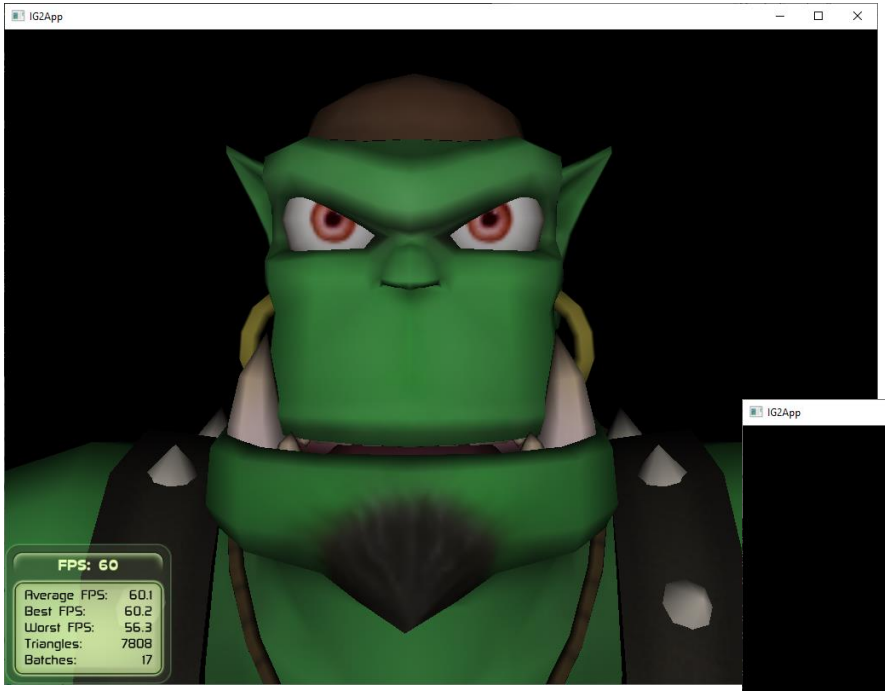
```
mCamMgr->setTarget(SceneNode*);
```

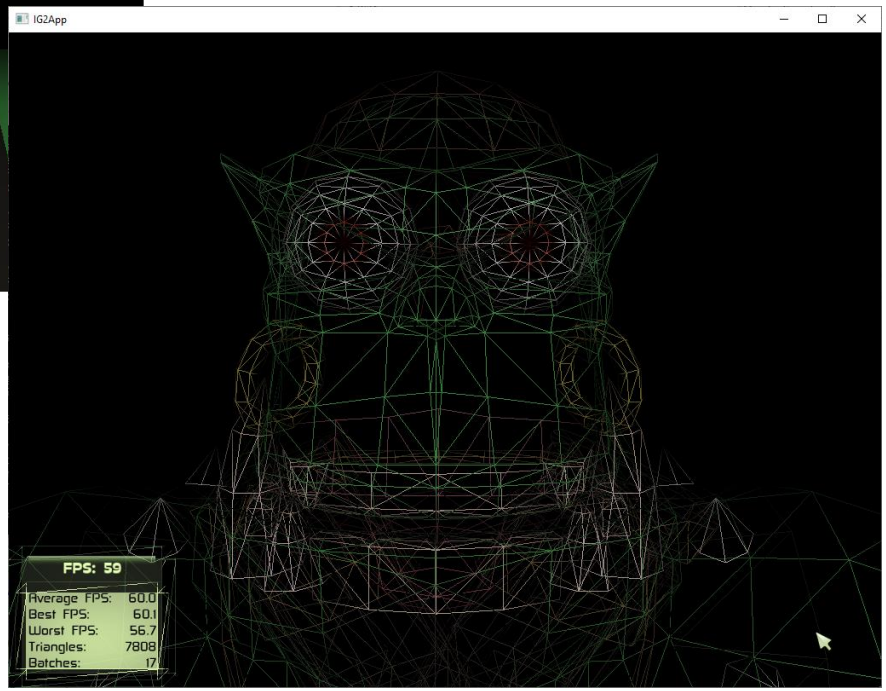
- Por defecto sigue al nodo raíz del grafo de la escena

```
mCamMgr = new OgreBites::CameraMan(mCamNode);

addInputListener(mCamMgr);

mCamMgr->setStyle(OgreBites::CS_ORBIT);
```

  - Muy útil al principio para entender los objetos de la escena
  - Es necesario crear previamente la cámara
    - El parámetro que recibe el constructor es un SceneNode
    - El SceneNode tiene la cámara adjunta (ver slide anterior)

cam->setPolygonMode(PM_POINTS|PM_WIREFRAME|PM_SOLID)





La luz

- La clase **Light** hereda de **MovableObject**.
- Las fuentes de luz **las crea el gestor de la escena**.

```
Light* luz = mSM->createLight("Luz");
lightNode->attachObject(luz);
mSM->setAmbientLight(ColourValue);   // Luz ambiente de la escena
```

- Las fuentes de luz en Ogre tienen componente difusa y especular
- La componente ambiente es general de la escena y la fija el gestor de escena
- Tenemos tres tipos de fuentes de luz (**LightTypes**):
  - LT_POINT
  - LT_SPOTLIGHT
  - LT_DIRECTIONAL

La luz

- Parte del código de **`setupScene()`** de **IG2App** para inicializar la luz

```
Light* luz = mSM->createLight("Luz");
luz->setType(Ogre::Light::LT_DIRECTIONAL);
luz->setDiffuseColour(0.75, 0.75, 0.75);

mLightNode = mSM->getRootSceneNode()->createChildSceneNode("nLuz");
mLightNode->attachObject(luz);

mLightNode->setDirection(Ogre::Vector3(0, 0, -1));
```

La luz

- Métodos heredados de **MovableObject**

```
light->setVisible(bool)  //false->apagar la luz
```

- Métodos de configuración de la luz a través del nodo o de la luz

```
lightNode->setDirection(Vec3); //luz direccional
light->setPosition(Vec3);      //luz posicional
```

- Métodos de configuración de la luz a través de la luz

```
light->setType(LightTypes);
light->setDiffuseColour(ColourValue);
light->setSpecularColour(ColourValue);
light->setAttenuation(…);
```

La luz

- Luz de tipo `LT_POINT`
  - Emiten luz en todas direcciones desde un punto en el espacio
  - Tienen intensidad y atenuación
  - Tienen un rango sobre el cual se aplica la iluminación
    - Fuera de este rango, los objetos no reciben luz
- Luz de tipo `LT_SPOTLIGHT`
  - Similares a las `LT_POINT` pero con dirección.
  - Tiene efecto cono
  - Como en las `LT_POINT`, tienen intensidad, atenuación y su efecto sobre un rango.
- Luz de tipo `LT_DIRECTIONAL`
  - No tiene intensidad en la atenuación (*falloff*)
  - Moverla no causaría efecto
  - No tienen posición específica.
    - P.ej: El sol.
  - Sí es posible modificar su orientación

Ejemplos de luz (El entorno)

- Crearemos un entorno de prueba para ver los distintos tipos de luz
  - Plano con una textura de piedras (`BeachStones`)
  - Malla `ogrehead`

```
MeshManager::getSingleton().createPlane("floor", ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
                                        Plane(Vector3::UNIT_Y, 0),
                                        1500, 1500, 200, 200, true, 1, 5, 5, Vector3::UNIT_Z);
    // Creating the floor
    Entity* ent = mSM->createEntity ("exampleFloor", "floor");
    ent->setMaterialName("example/stonesFloor");
    SceneNode* floor = mSM->getRootSceneNode()->createChildSceneNode();
    floor->attachObject(ent);

    // Creating ogrehead (the villain)
    Ogre::Entity * ogreheadEnt = mSM->createEntity("ogrehead.mesh");
    mOgreheadNode =  mSM->getRootSceneNode()->createChildSceneNode();
    mOgreheadNode->attachObject(ogreheadEnt);
    mOgreheadNode->scale(0.7, 0.7, 0.7);
    mOgreheadNode->setPosition(0, 20, 0);
```

Ejemplo de luz (Point light)

```
// The light
Ogre::Light* pointLight1 = mSM->createLight("PointLight1");
pointLight1->setType(Light::LT_POINT);
pointLight1->setDiffuseColour(1.0f,1.0f,1.0f);

// Node with the light attached
nodePoint = mSM->getRootSceneNode()->createChildSceneNode();
nodePoint->setPosition(50, 30, 30);
nodePoint->attachObject(pointLight1);
```

Ejemplo de luz (Point light)

```
// The light
Ogre::Light* pointLight1 = mSM->createLight("PointLight1");
pointLight1->setType(Light::LT_POINT);
pointLight1->setDiffuseColour(0.0f,0.0f,1.0f);   // Cambiamos de luz blanca a luz azul

// Node with the light attached
nodePoint = mSM->getRootSceneNode()->createChildSceneNode();
nodePoint->setPosition(50, 30, 30);
nodePoint->attachObject(pointLight1);
```

Spotlights

- Efecto del cono de luz:

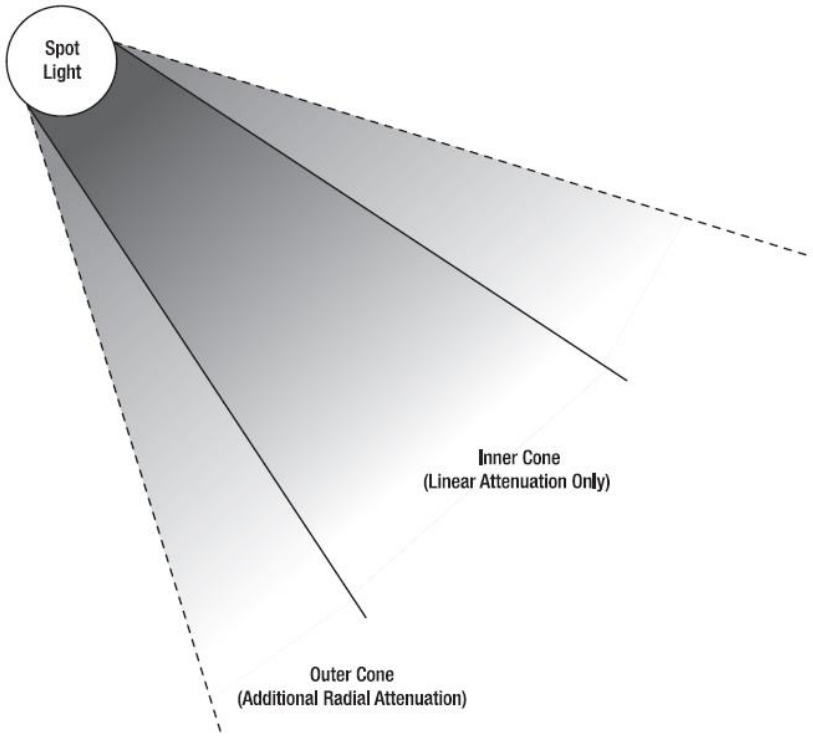
```
spotLight->setSpotlightRange(innerAngle, outerAngle, falloff=1.0);
```

  - El cono interior se aplica solo en Direct3D
  - En OpenGL vale 0.

El ratio del parámetro **falloff** (atenuación) entre los conos interno y externo.

Valores de f*alloff:*

- 1.0 indica un falloff lineal
- <1.0 indica un fallow más lento
- >1.0 un falloff más rápido.



Fuente: Pro OGRE 3D Programming. Gregory Junker

Spotlights

- Configuración de un foco:

```
luzFoco = mSM->createLight("Luz Foco");
luzFoco->setType(Ogre::Light::LT_SPOTLIGHT);
luzFoco->setDiffuseColour(Ogre::ColourValue(1.0f,1.0f,1.0f));

luzFoco->setDirection(Ogre::Vector3(1, -1, 0));
luzFoco->setSpotlightInnerAngle(Ogre::Degree(5.0f));
luzFoco->setSpotlightOuterAngle(Ogre::Degree(45.0f));
luzFoco->setSpotlightFalloff(0.0f);

node->attachObject(luzFoco);
```

Ejemplo de luz (Spotlight)

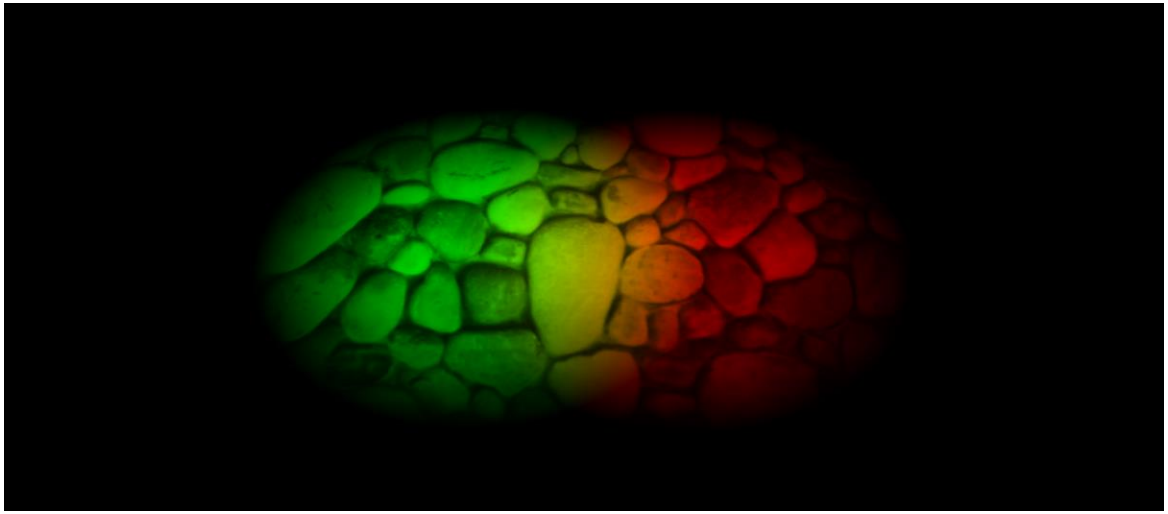
```
// The light
Ogre::Light* spotLight1 = mSM->createLight("SpotLight1");
spotLight1->setType(Light::LT_SPOTLIGHT);
spotLight1->setSpotlightInnerAngle(Ogre::Degree(5.0f));
spotLight1->setSpotlightOuterAngle(Ogre::Degree(45.0f));
spotLight1->setSpotlightFalloff(0.0f);
spotLight1->setDiffuseColour(1.0f,1.0f,1.0f);

// Node with the light attached
nodeSpot1 = mSM->getRootSceneNode()->createChildSceneNode();
nodeSpot1->setPosition(100,100,100);
nodeSpot1->setDirection(Ogre::Vector3(-1,-1,-1));
nodeSpot1->attachObject(spotLight1);
```

Ejemplo de luz (Spotlight)

```
// The light
Ogre::Light* spotLight1 = mSM->createLight("SpotLight1");
spotLight1->setType(Light::LT_SPOTLIGHT);
spotLight1->setSpotlightInnerAngle(Ogre::Degree(5.0f));
spotLight1->setSpotlightOuterAngle(Ogre::Degree(45.0f));
spotLight1->setSpotlightFalloff(0.0f);
spotLight1->setDiffuseColour(1.0f,0.0f,0.0f);        // Cambiamos de luz blanca a luz roja

// Node with the light attached
nodeSpot1 = mSM->getRootSceneNode()->createChildSceneNode();
nodeSpot1->setPosition(100,100,100);
nodeSpot1->setDirection(Ogre::Vector3(-1,-1,-1));
nodeSpot1->attachObject(spotLight1);
```

Ejemplo de luz (Spotlight)

```
// The lights
spotLight1->setDiffuseColour(1.0f, 0.0f, 0.0f);                    // Luz roja
nodeSpot1 = mSM->getRootSceneNode()->createChildSceneNode();
nodeSpot1->setPosition(50,200,0);
nodeSpot1->setDirection(Ogre::Vector3(0,-1,0));
spotLight2->setDiffuseColour(0.0f, 1.0f, 0.0f);                    // Luz verde
nodeSpot2->setPosition(-50,200,0);
nodeSpot2->setDirection(Ogre::Vector3(0,-1,0));
```



Ejemplo de luz (Directional)

```
// The light
Ogre::Light* directionalLight1 = mSM->createLight("DirectionalLight1");
directionalLight1->setType(Light::LT_DIRECTIONAL);
directionalLight1->setDiffuseColour(1.0f,1.0f,1.0f);

// Node with the light attached
nodeDir = mSM->getRootSceneNode()->createChildSceneNode();
nodeDir->setDirection(Ogre::Vector3(0,-1,0));
nodeDir->attachObject(directionalLight1);
```

La luz: Sombras

- En esencia, OGRE proporciona dos formas de implementar sombras
  - Stencil (troquel/plantilla)
  - Basadas en texturas
- Todas las luces producen sombra por defecto.
- Para indicar el tipo de sombra, se indica a través del scene manager:

```
mSM->setShadowTechnique(Ogre::ShadowTechnique);
```

  - Algunos ejemplos:
    - SHADOWDETAILTYPE_STENCIL
    - SHADOWDETAILTYPE_TEXTURE
    - SHADOWTYPE_STENCIL_ADDITIVE
    - SHADOWTYPE_TEXTURE_ADDITIVE
    - SHADOWTYPE_TEXTURE_ADDITIVE_INTEGRATED
    - SHADOWTYPE_TEXTURE_MODULATIVE_INTEGRATED
- Cada luz puede activar/desactivar la producción de sombras

```
light->setCastShadows(true|false);
```

Stencil shadows

- Método mediante el cual se crea una "máscara" para la pantalla utilizando la técnica denominada "stencil buffer".
- Esta máscara puede utilizarse para excluir zonas de la pantalla de posteriores renderizaciones
  - Puede utilizarse para incluir o excluir zonas en sombra
  - Parámetros: `SHADOWTYPE_STENCIL_ADDITIVE` o `SHADOWTYPE_STENCIL_MODULATIVE`
- Para generar el stencil (plantilla o troquel), los "volúmenes de sombra" se renderizan extruyendo la silueta del emisor de la sombra lejos de la luz.
- La ventaja de estas sombras es que pueden hacer auto-sombreado de forma sencilla en hardware de gama baja.
  - Siempre que se utilice un **número moderado** de polígonos.
- Sin embargo, existen numerosas desventajas (especialmente en hardware más moderno)
- Dado que son una técnica geométrica, son más costosas cuanto mayor sea el número de polígonos
  - Penaliza significativamente al aumentar el detalle de las mallas.

Texture-based shadows

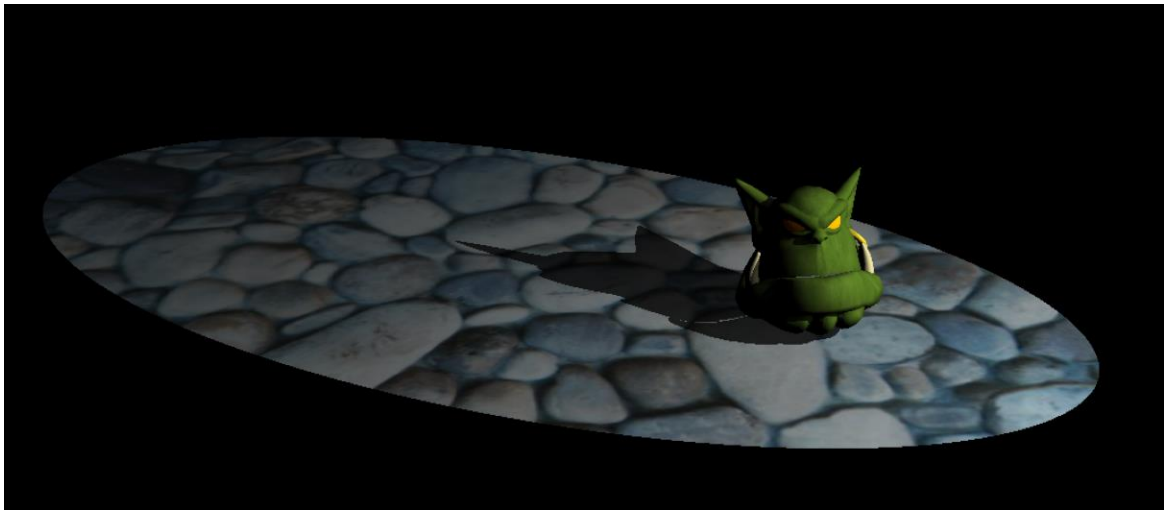
- Consisten en renderizar los emisores de sombras **desde el punto de vista de la luz en una textura**, que luego se proyecta sobre los receptores de sombras.
- Su principal ventaja - frente a las *stencil shadows* - es que la sobrecarga de aumentar el detalle geométrico es mucho menor
  - No es necesario realizar cálculos por triángulo
  - La mayor parte del trabajo de renderizado lo realiza la tarjeta gráfica
- Son mucho más personalizables
  - Pueden ser introducidas en *shadders* para aplicarlas como requiera la escena
    - Es possible realizar filtrados para crear sombras más suaves o aplicarles otros efectos
- La mayoría de los motores modernos utilizan esta técnica de sombreado
- Su principal desventaja es que, al ser simplemente una textura, tienen una resolución fija
  - Si se estiran, la "pixelación" de la textura puede hacerse evidente.

Ejemplo de sombra (Stencil shadow)

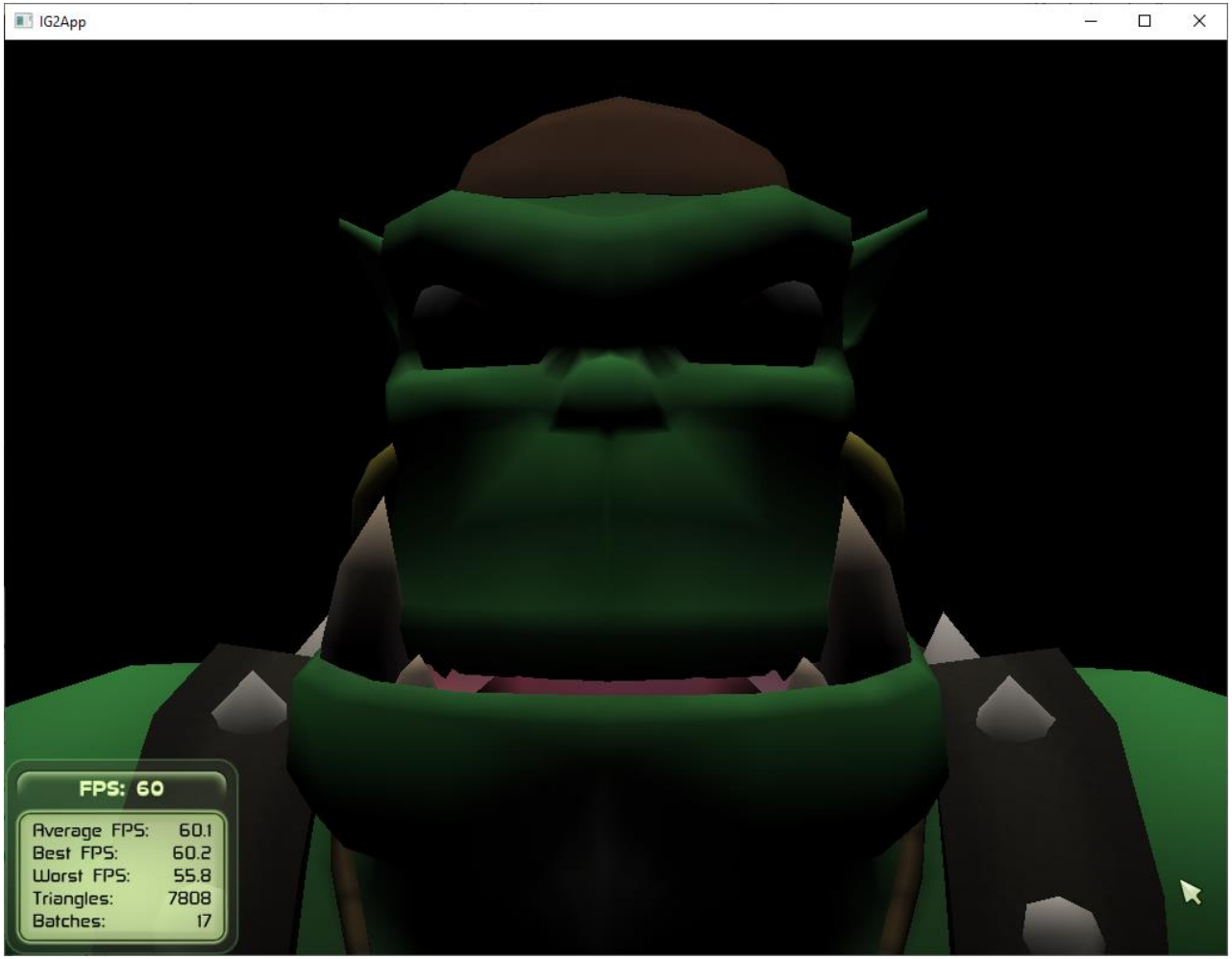
```
// Shadows type
mSM->setShadowTechnique(Ogre::SHADOWTYPE_STENCIL_ADDITIVE);

// The light
Ogre::Light* spotLight1 = mSM->createLight("SpotLight1");
spotLight1->setType(Light::LT_SPOTLIGHT);
spotLight1->setDiffuseColour(1.0f, 1.0f, 1.0f);

// Node with the light attached
nodeSpot1 = mSM->getRootSceneNode()->createChildSceneNode();
nodeSpot1->setPosition(100,100,100);
nodeSpot1->setDirection(Ogre::Vector3(-1,-1,-1));
 nodeSpot1->attachObject(spotLight1);
```

Ejemplo de sombra (Stencil shadow)

```
// Shadows type
mSM->setShadowTechnique(Ogre::SHADOWTYPE_STENCIL_MODULATIVE);

// The light
Ogre::Light* spotLight1 = mSM->createLight("SpotLight1");
spotLight1->setType(Light::LT_SPOTLIGHT);
spotLight1->setDiffuseColour(1.0f, 1.0f, 1.0f);

// Node with the light attached
nodeSpot1 = mSM->getRootSceneNode()->createChildSceneNode();
nodeSpot1->setPosition(100,100,100);
nodeSpot1->setDirection(Ogre::Vector3(-1,-1,-1));
 nodeSpot1->attachObject(spotLight1);
```



La luz


IG2App
FPS: 60
Average FPS: 60.1
Best FPS: 60.2
Worst FPS: 55.8
Triangles: 7808
Batches: 17