Table of contents

This is a small code snippet, a small solution to an old issue in Ogre: The lack of a simple way of drawing 2D objects to the screen, the old way. Maybe for programming a -HUD, maybe for doing a small mini game inside a bigger 3D game, maybe just for using Ogre for 2D programming.

-Overlays are nice and -CEGUI is nice, but they aren't versatile enough if you want to program, say, a Tetris or a Mario game. Not that Ogre doesn't allow you to program in 2D using 3D primitives, but you have to use the hardware buffers and that's a lot of hard work.

The code snippet I present here allows you to blit sprites to screen, just as you did in the old times of DirectDraw or even before, in the days of memory copying for DOS games.

For Mogre a port is available in MOGRE SpriteManager2d.

The original thread is here

ogre2d-main.h

First, the code and then some extra information about its use.

```
// Ogre 2d: a small wrapper for 2d Graphics Programming in Ogre.
/*
   Wrapper for 2d Graphics in the Ogre 3d engine.

   Coded by H. HernÃ¡n Moraldo from Moraldo Games
   www.hernan.moraldo.com.ar/pmenglish/field.php

   Thanks for the Cegui team as their rendering code in Ogre gave me
   fundamental insight on the management of hardware buffers in Ogre.

   --------------------

   Copyright (c) 2006 Horacio Hernan Moraldo

   This software is provided 'as-is', without any express or
   implied warranty. In no event will the authors be held liable
   for any damages arising from the use of this software.

   Permission is granted to anyone to use this software for any
   purpose, including commercial applications, and to alter it and
   redistribute it freely, subject to the following restrictions:

   1. The origin of this software must not be misrepresented; you
   must not claim that you wrote the original software. If you use
   this software in a product, an acknowledgment in the product
   documentation would be appreciated but is not required.

   2. Altered source versions must be plainly marked as such, and
   must not be misrepresented as being the original software.

   3. This notice may not be removed or altered from any source
   distribution.

*/
#ifndef __OGRE2D_MAIN_FILE
#define __OGRE2D_MAIN_FILE

#include <Ogre.h>
#include <OgreRenderQueueListener.h>

#include <string>
#include <list>

struct Ogre2dSprite
{
   double x1, y1, x2, y2;// sprite coordinates
   double tx1, ty1, tx2, ty2;// texture coordinates
   Ogre::ResourceHandle texHandle;// texture handle
};

struct VertexChunk {
   Ogre::ResourceHandle texHandle;
   unsigned int vertexCount;
};

class Ogre2dManager:public Ogre::RenderQueueListener
```

```
{
private:
   Ogre::SceneManager* sceneMan;

   Ogre::uint8 targetQueue;
   bool afterQueue;
public:
   Ogre2dManager();
   virtual ~Ogre2dManager();

   /// Initializes this 2d Manager
   /** and registers it as the render queue listener.*/
   void init(Ogre::SceneManager* sceneMan, Ogre::uint8 targetQueue, bool afterQueue);
   /// Finishes Ogre 2d Manager
   void end();

   /// Called by Ogre, for being a render queue listener
   virtual void renderQueueStarted(
      Ogre::uint8 queueGroupId, const Ogre::String &invocation, bool &skipThisInvocation);
   /// Called by Ogre, for being a render queue listener
   virtual void renderQueueEnded(
      Ogre::uint8 queueGroupId, const Ogre::String &invocation, bool &repeatThisInvocation);

   /// Buffers a sprite to be sent to the screen at render time.
   /**
      Sprite coordinates are in screen space: top left pixel is (-1, 1), and bottom right
      is (1, -1). The texture space, instead, ranges from (0, 0) to (1, 1).

      /param textureName Name of the texture to use in this sprite (remember: texture
      name, not material name!). The texture has to be already loaded by Ogre for this
      to work.
      /param x1 x coordinate for the top left point in the sprite.
      /param y1 y coordinate for the top left point in the sprite.
      /param x2 x coordinate for the bottom right point in the sprite.
      /param y2 y coordinate for the bottom right point in the sprite.
      /param tx1 u coordinate for the texture, in the top left point of the sprite.
      /param ty1 v coordinate for the texture, in the top left point of the sprite.
      /param tx2 u coordinate for the texture, in the bottom right point of the sprite.
      /param ty2 u coordinate for the texture, in the bottom right point of the sprite.
   */
   void spriteBltFull(std::string textureName, double x1, double y1, double x2, double y2, do
uble tx1=0, double ty1=0, double tx2=1, double ty2=1);

private:
   /// Render all the 2d data stored in the hardware buffers.
   void renderBuffer();
   /// Create a new hardware buffer
   /**
   /param size Vertex count for the new hardware buffer.
   */
   void createHardwareBuffer(unsigned int size);
   /// Destroy the hardware buffer
   void destroyHardwareBuffer();
   /// Set Ogre for rendering
   void prepareForRender();

   // ogre specifics
```

```
    Ogre::RenderOperation renderOp;
    Ogre::HardwareVertexBufferSharedPtr hardwareBuffer;

    // sprite buffer
    std::list<Ogre2dSprite> sprites;
};

#endif // __OGRE2D_MAIN_FILE
```

ogre2d-main.cpp

```
// Ogre 2d: a small wrapper for 2d Graphics Programming in Ogre.
/*
   Wrapper for 2d Graphics in the Ogre 3d engine.

   Coded by H. HernÃ¡n Moraldo from Moraldo Games
   www.hernan.moraldo.com.ar/pmenglish/field.php

   Thanks for the Cegui team as their rendering code in Ogre gave me
   fundamental insight on the management of hardware buffers in Ogre.

   --------------------

   Copyright (c) 2006 Horacio Hernan Moraldo

   This software is provided 'as-is', without any express or
   implied warranty. In no event will the authors be held liable
   for any damages arising from the use of this software.

   Permission is granted to anyone to use this software for any
   purpose, including commercial applications, and to alter it and
   redistribute it freely, subject to the following restrictions:

   1. The origin of this software must not be misrepresented; you
   must not claim that you wrote the original software. If you use
   this software in a product, an acknowledgment in the product
   documentation would be appreciated but is not required.

   2. Altered source versions must be plainly marked as such, and
   must not be misrepresented as being the original software.

   3. This notice may not be removed or altered from any source
   distribution.

*/
#include "ogre2d-main.h"
#include <Ogre.h>
#include <OgreMesh.h>
#include <OgreHardwareBuffer.h>

#define OGRE2D_MINIMAL_HARDWARE_BUFFER_SIZE 120

Ogre2dManager::Ogre2dManager()
{
}

Ogre2dManager::~Ogre2dManager()
{
}

void Ogre2dManager::init(Ogre::SceneManager* sceneMan, Ogre::uint8 targetQueue, bool afterQue
ue)
{
   Ogre2dManager::sceneMan=sceneMan;
   Ogre2dManager::afterQueue=afterQueue;
   Ogre2dManager::targetQueue=targetQueue;
```

```
   hardwareBuffer.setNull();

   sceneMan->addRenderQueueListener(this);
}

void Ogre2dManager::end()
{
   if (!hardwareBuffer.isNull())
      destroyHardwareBuffer();

   sceneMan->removeRenderQueueListener(this);
}

void Ogre2dManager::renderQueueStarted(
   Ogre::uint8 queueGroupId, const Ogre::String &invocation, bool &skipThisInvocation)
{
   if (!afterQueue && queueGroupId==targetQueue)
      renderBuffer();
}

void Ogre2dManager::renderQueueEnded(
   Ogre::uint8 queueGroupId, const Ogre::String &invocation, bool &repeatThisInvocation)
{
   if (afterQueue && queueGroupId==targetQueue)
      renderBuffer();
}

void Ogre2dManager::renderBuffer()
{
   Ogre::RenderSystem* rs=Ogre::Root::getSingleton().getRenderSystem();
   std::list<Ogre2dSprite>::iterator currSpr, endSpr;

   VertexChunk thisChunk;
   std::list<VertexChunk> chunks;

   unsigned int newSize;

   newSize=sprites.size()*6;
   if (newSize<OGRE2D_MINIMAL_HARDWARE_BUFFER_SIZE)
      newSize=OGRE2D_MINIMAL_HARDWARE_BUFFER_SIZE;

   // grow hardware buffer if needed
   if (hardwareBuffer.isNull() || hardwareBuffer->getNumVertices()<newSize)
   {
      if (!hardwareBuffer.isNull())
         destroyHardwareBuffer();

      createHardwareBuffer(newSize);
   }

   if (sprites.empty()) return;

   // write quads to the hardware buffer, and remember chunks
   float* buffer;
   float z=-1;

   buffer=(float*)hardwareBuffer->lock(Ogre::HardwareBuffer::HBL_DISCARD);
```

```
endSpr=sprites.end();
currSpr=sprites.begin();
thisChunk.texHandle=currSpr->texHandle;
thisChunk.vertexCount=0;
while (currSpr!=endSpr)
{
   // 1st point (left bottom)
   *buffer=currSpr->x1; buffer++;
   *buffer=currSpr->y2; buffer++;
   *buffer=z; buffer++;
   *buffer=currSpr->tx1; buffer++;
   *buffer=currSpr->ty2; buffer++;
   // 2st point (right top)
   *buffer=currSpr->x2; buffer++;
   *buffer=currSpr->y1; buffer++;
   *buffer=z; buffer++;
   *buffer=currSpr->tx2; buffer++;
   *buffer=currSpr->ty1; buffer++;
   // 3rd point (left top)
   *buffer=currSpr->x1; buffer++;
   *buffer=currSpr->y1; buffer++;
   *buffer=z; buffer++;
   *buffer=currSpr->tx1; buffer++;
   *buffer=currSpr->ty1; buffer++;

   // 4th point (left bottom)
   *buffer=currSpr->x1; buffer++;
   *buffer=currSpr->y2; buffer++;
   *buffer=z; buffer++;
   *buffer=currSpr->tx1; buffer++;
   *buffer=currSpr->ty2; buffer++;
   // 5th point (right bottom)
   *buffer=currSpr->x2; buffer++;
   *buffer=currSpr->y1; buffer++;
   *buffer=z; buffer++;
   *buffer=currSpr->tx2; buffer++;
   *buffer=currSpr->ty1; buffer++;
   // 6th point (right top)
   *buffer=currSpr->x2; buffer++;
   *buffer=currSpr->y2; buffer++;
   *buffer=z; buffer++;
   *buffer=currSpr->tx2; buffer++;
   *buffer=currSpr->ty2; buffer++;

   // remember this chunk
   thisChunk.vertexCount+=6;
   currSpr++;
   if (currSpr==endSpr || thisChunk.texHandle!=currSpr->texHandle)
   {
      chunks.push_back(thisChunk);
      if (currSpr!=endSpr)
      {
         thisChunk.texHandle=currSpr->texHandle;
         thisChunk.vertexCount=0;
      }
   }
```

```
    }

    hardwareBuffer->unlock();

    // set up...
    prepareForRender();

    // do the real render!
    Ogre::TexturePtr tp;
    std::list<VertexChunk>::iterator currChunk, endChunk;

    endChunk=chunks.end();
    renderOp.vertexData->vertexStart=0;
    for (currChunk=chunks.begin(); currChunk!=endChunk; currChunk++)
    {
       renderOp.vertexData->vertexCount=currChunk->vertexCount;
       tp=Ogre::TextureManager::getSingleton().getByHandle(currChunk->texHandle);
       rs->_setTexture(0, true, tp->getName());
       rs->_render(renderOp);
       renderOp.vertexData->vertexStart+=currChunk->vertexCount;
    }

    // sprites go home!
    sprites.clear();
}

void Ogre2dManager::prepareForRender()
{
    Ogre::LayerBlendModeEx colorBlendMode;
    Ogre::LayerBlendModeEx alphaBlendMode;
    Ogre::TextureUnitState::UVWAddressingMode uvwAddressMode;

    Ogre::RenderSystem* rs=Ogre::Root::getSingleton().getRenderSystem();

    colorBlendMode.blendType=Ogre::LBT_COLOUR;
    colorBlendMode.source1=Ogre::LBS_TEXTURE;
    colorBlendMode.operation=Ogre::LBX_SOURCE1;

    alphaBlendMode.blendType=Ogre::LBT_ALPHA;
    alphaBlendMode.source1=Ogre::LBS_TEXTURE;
    alphaBlendMode.operation=Ogre::LBX_SOURCE1;

    uvwAddressMode.u=Ogre::TextureUnitState::TAM_CLAMP;
    uvwAddressMode.v=Ogre::TextureUnitState::TAM_CLAMP;
    uvwAddressMode.w=Ogre::TextureUnitState::TAM_CLAMP;

    rs->_setWorldMatrix(Ogre::Matrix4::IDENTITY);
    rs->_setViewMatrix(Ogre::Matrix4::IDENTITY);
    rs->_setProjectionMatrix(Ogre::Matrix4::IDENTITY);
    rs->_setTextureMatrix(0, Ogre::Matrix4::IDENTITY);
    rs->_setTextureCoordSet(0, 0);
    rs->_setTextureCoordCalculation(0, Ogre::TEXCALC_NONE);
    rs->_setTextureUnitFiltering(0, Ogre::FO_LINEAR, Ogre::FO_LINEAR, Ogre::FO_POINT);
    rs->_setTextureBlendMode(0, colorBlendMode);
    rs->_setTextureBlendMode(0, alphaBlendMode);
    rs->_setTextureAddressingMode(0, uvwAddressMode);
    rs->_disableTextureUnitsFrom(1);
```

```
   rs->setLightingEnabled(false);
   rs->_setFog(Ogre::FOG_NONE);
   rs->_setCullingMode(Ogre::CULL_NONE);
   rs->_setDepthBufferParams(false, false);
   rs->_setColourBufferWriteEnabled(true, true, true, false);
   rs->setShadingType(Ogre::SO_GOURAUD);
   rs->_setPolygonMode(Ogre::PM_SOLID);
   rs->unbindGpuProgram(Ogre::GPT_FRAGMENT_PROGRAM);
   rs->unbindGpuProgram(Ogre::GPT_VERTEX_PROGRAM);
   rs->_setSceneBlending(Ogre::SBF_SOURCE_ALPHA, Ogre::SBF_ONE_MINUS_SOURCE_ALPHA);
   rs->_setAlphaRejectSettings(Ogre::CMPF_ALWAYS_PASS, 0);
}

void Ogre2dManager::createHardwareBuffer(unsigned int size)
{
   Ogre::VertexDeclaration* vd;

   renderOp.vertexData=new Ogre::VertexData;
   renderOp.vertexData->vertexStart=0;

   vd=renderOp.vertexData->vertexDeclaration;
   vd->addElement(0, 0, Ogre::VET_FLOAT3, Ogre::VES_POSITION);
   vd->addElement(0, Ogre::VertexElement::getTypeSize(Ogre::VET_FLOAT3),
      Ogre::VET_FLOAT2, Ogre::VES_TEXTURE_COORDINATES);

   hardwareBuffer=Ogre::HardwareBufferManager::getSingleton().createVertexBuffer(
      vd->getVertexSize(0),
      size,// buffer size
      Ogre::HardwareBuffer::HBU_DYNAMIC_WRITE_ONLY_DISCARDABLE,
      false);// use shadow buffer? no

   renderOp.vertexData->vertexBufferBinding->setBinding(0, hardwareBuffer);

   renderOp.operationType=Ogre::RenderOperation::OT_TRIANGLE_LIST;
   renderOp.useIndexes=false;

}

void Ogre2dManager::destroyHardwareBuffer()
{
   delete renderOp.vertexData;
   renderOp.vertexData=0;
   hardwareBuffer.setNull();
}

void Ogre2dManager::spriteBltFull(
   std::string textureName,
   double x1, double y1, double x2, double y2,
   double tx1, double ty1, double tx2, double ty2)
{
   Ogre::TexturePtr tp;
   Ogre2dSprite spr;

   spr.x1=x1;
   spr.y1=y1;
   spr.x2=x2;
   spr.y2=y2;
```

```
    spr.tx1=tx1;
    spr.ty1=ty1;
    spr.tx2=tx2;
    spr.ty2=ty2;

    tp=Ogre::TextureManager::getSingleton().getByName(textureName);
    spr.texHandle=tp->getHandle();

    sprites.push_back(spr);
}
```

Notes

Some important things about this code:

- Initialize the wrapper in your setup code, ie:

```
Ogre2dManager* ogre2dManager=new Ogre2dManager;
ogre2dManager->init(mSceneMgr, Ogre::RENDER_QUEUE_OVERLAY, true);
```

- End the wrapper when you are not going to use it anymore

```
ogre2dManager->end();
delete ogre2dManager;
```

- Once per frame, at frame start, you can use sprite blits to draw 2D figures on the screen... actually those are buffered and - at render time - they are sent to the hardware buffers and rendered. Of course, the buffer is cleared as soon as it's rendered to the screen... don't expect sprites to be persistent as meshes are in Ogre, sprites are not! You decide what to blit to the screen every frame and each time it can be a different amount of sprites.

```
ogre2dManager->spriteBltFull("myjpg1.jpg", -0.5, 0.5, 0.5, -0.5);
ogre2dManager->spriteBltFull("mytexture.jpg", -0.3, 0.3, 0.3, -0.3);
ogre2dManager->spriteBltFull("mytexture.jpg", 0.3, 0.2, 0.7, -0.2);
ogre2dManager->spriteBltFull("othertexture.jpg", -0.1, 0.1, 0.1, -0.1);
```

- The textures you use for your sprites have to be already loaded in the Ogre system to work. This is important! Otherwise you'll be warned with an exception.
- If you are having trouble getting this code to work using Eihort try loading your textures like this:

```
Ogre::TextureManager::getSingleton().load("Image.jpg",ResourceGroupManager::DEFAULT_RESOURCE_
GROUP_NAME);
```

and unloading your texture like this:

```
Ogre::TextureManager::getSingleton().unload("Image.jpg");
```

Contributors to this page: jacmoe and Spacegaier .
Page last modified on Friday 01 of January, 2010 20:15:11 -08 by jacmoe.

Source | History

Search by Tags

Search Wiki by Freetags

Latest Changes

1. AMOFGameEngine
2. MOGRE
3. Roadmap
4. Home
5. Easy Ogre Exporter
6. OGRE Exporters
7. Light mapping
8. Ogre 2.1 FAQ
9. ManualObject
10. Advanced Mogre Framework

...more

Search

Find

Go

Advanced Search Help

Online Users

16 online users