I-Cube 3D Engine User Manual

v1.1

http://www.simviu.com/dev

Overview

I-Cube 3D Engine (**IcEng**) is an open source, light weight OpenGL/C++ 3D graphics Engine. It's designed with simple structure and interface to fit small cross platform 3D application. It can also serve as OpenGL example, or foundation code base that you can develop your own 3D engine upon.

The source or binary are provided with free BSD license, so you can use this engine on personal or commercial project with no restriction. See **License** section.

For more information check website http://www.simviu.com/dev.

Features:

- C++ 0x11 code, OpenGL 2.0 shader.
- Cross Platform (Mac/Windows/Linux/iOS/Android), project file provided.
- Auto memory and resource management with C++ 0x11 std::shared_ptr.
- Model loading with Waveform .OBJ file. (More format in future)
- Basic lighting, material.
- Abstract C++ wrapper class with OpenGL function hidden.

Architecture

The IcEng applications are wrapped in **IcApp**/**IcWindow**/**IcScene** structure.

- There is only one **IcApp** instance(singleton).
- An IcApp application can contain several **IcWindow**.
- Each window has it's own OpenGL context.
- Each window has one or more **IcScene**.
- Each **IcScene** contain one **IcCamera**, and many **IcObject**.
- An **IcObject** has it's **IcModel**, loaded from **OBJ** file or created dynamically.
- **IcModel** are constructed by **IcMesh**, **IcTexture** and **IcMaterial**.

In most case your simple 3d application has one **IcWindow** and one **IcScene**. For iOS/Android mobile App, there is only one window.

Quick Start

The easiest example is inside IcEng/src/test/MyTestApp.cpp. Here is the detail of this example.

Step1: Create Scene

The I-Cube 3D Application start with overriding **IcScene** class, and override virtual function **onInit()** to initialize a 3D scene, and **onUpdate(double deltaT)** for animation .

```
#include "MyTestApp.h"

using namespace Ic3d;
using namespace ctl;
using namespace std;

//-----------------------------
// MyTestScn
//-----------------------------
class MyTestScn : public IcScene
{
protected:
    Sp<IcObject> m_pObj = nullptr;
    float m_degree = 0; // rotation degree

public:
    //--------------------------
    //  onInit
    //--------------------------
    void onInit() override
    {
        IcScene::onInit();
        logInfo("MyTestScn::onInit()");

        IcMeshData mshd;
        mshd.createCube(TVec3(1,1,1), TVec3(0,0,0));
        auto pModel = makeSp<IcModel>(mshd);
        auto pObj   = makeSp<IcObject>(pModel);
        addObj(pObj);
        auto& cam = *getCamera();
        cam.setPos(TVec3(10, 4, -6));
        cam.lookAt(TVec3(0,0,0), TVec3(0,1,0));
        m_pObj = pObj;
    };
    //--------------------------
    //  onUpdate
    //--------------------------
    void onUpdate(double deltaT) override
```

```
    {
        static float K_rotSpeed = 30.0;
        float dt = deltaT > 1.0 ? 1.0 : deltaT;
        if(m_pObj==nullptr) return;
        m_degree += K_rotSpeed*dt;
        if(m_degree>360) m_degree -= 360;
        TQuat q(TVec3(0, deg2rad(m_degree), 0));
        m_pObj->setQuat(q);
    }
};
```

The above code initialize a 3D scene, create **IcModel** of cube, then create **IcObject** with this model and add to scene. In **onUpdate()** function it rotate the camera.

The template function `ctl::makeSp<class>` , `ctl::Sp<class>` is aliase to **make_shared** and **shared_ptr**, provided by C++ 0x11. The memory of shared pointers are automatically managed, meaning it's not required to be deleted manually. When all the references of one pointer is out of scope, it will be automatically deleted. So there is no **new**/**delete** pair in Ic3d code. This is very convenient for 3D applications, as lots of resource sharing happen.

The aliases of **makeSp**/**Sp** is for in case you want to replace it by other memory management library such as **BOOST**. The aliases are defined in **ctl.h**, where you can easily modify. If you replace them with standard C++ **new**, you need to handle deletion manually to avoid memory leak.

Step2: Create window and app

Once the 3D scenery are established, this scene can be put into a **IcWindow** and **IcApp** easily.

```
//-----------------------------
// MyTestWindow
//-----------------------------
class MyTestWindow : public IcWindow
{
public:
    virtual  void onInit() override
    {
        IcWindow::onInit();
        logInfo("MyTestWindow::onInit()");
        addScene(makeSp<MyTestScn>());
    }
};
//-----------------------------
// MyTestApp
//-----------------------------
void MyTestApp::onInit()
{
    IcApp::onInit();
    logInfo("MyTestApp::onInit()");
    addWindow(makeSp<MyTestWindow>());
```

```
};
```

Note: For cross platform consistence, put none OpenGL initialization code in overrided **IcApp::onInit()**. The overrided **IcWindow::onInit()** is the place where OpenGL context are ready.

Step3: Launch App

Once your App is established, the last thing is to launch our app. It vary on deferent platforms. The purpose of **IcApp**/**IcWindow**/**IcScene** structure is to wrap all your functions and logic inside **IcApp**, thus maintain code inside overridden **IcApp** completely same cross platforms.

A) Launch App on PC(Mac/Linux/Windows)

Launch App on PC is by IcApp member function **runCmdLine(argc, argv)**, as below.

```
#include "MyTestApp.h"
//--------------------
//  main
//--------------------
int main(int argc, char **argv)
{
    MyTestApp app;
    return app.runCmdLine(argc, argv);
}
```

The default Ic3d window system is **GLUT**. So **GLUT** command line options are supported here. You can replace with other window system with your choice, such as **GLFW** or **WGL** (MS Windows). As long as it provide OpenGL 2.0 context. To do that you need to override default **IcWinMng** class, and set it's singleton instance.

B) Launch App on iOS

On iOS we provide OpenGL context by Object-C class **IcViewController**. First in your basic view controller instantiate your App.

```
#import "ViewController.h"
#import "IcViewController.h"
#import "MyTestApp.h"
@interface ViewController ()
{
    MyTestApp   m_myApp;
}
@end
```

Then some where in your view controller launch **IcViewController** and call **initIcApp()** with your app. The following code showed example of launch **IcViewController** by press a button.

```
- (IBAction)onButton_Test:(id)sender {
```

```
    IcViewController* vcon = [[IcViewController alloc] init];
    [vcon initIcApp:&m_myApp];
    [self presentViewController:vcon
          animated:NO completion:nil];
}
```

c) Launch App on Android

On android, JAVA code communicate with C++ code indirectly via **JNI** interface. In your **JNI** code, init app as showed below.

```
#include "MyTestApp.h"
#include "IcEngJNI.h"
#include "../../../../../../../src/test/MyTestApp.h"

static MyTestApp l_app;
extern "C" JNIEXPORT void JNICALL
  Java_com_simviu_IcEngTest_MainActivity_initIcApp
    (JNIEnv * env,  jobject obj,  jstring jsPathCache)
{
    std::string sPathCache =
          IcEngJNI::jstr2str(env, jsPathCache);
    IcEngJNI::initIcApp(&l_app, sPathCache, sPathCache);
}
```

The code above is JNI function can be called from JAVA class. The following code snap showed how JAVA class invoke JNI functions. Make sure the the prefix above match the package name of Java class.

```
static {
    System.loadLibrary("IcEngTest");
}
static native void initIcApp(String sPathCache);
```

Then in your very beginning of the app, usually in **onCreate()** function of your **MainActivity**, call **initIcApp()**, example below:

```
    //---- Create IcApp at very beginning
    String sPathCache = getCacheDir().toString();
    initIcApp(sPathCache);
    //----- Copy asset
    IcEngJNI.copyAssetDir(this, "IcShader");
```

The code above called JNI function initIcApp(), with parameter of cache dir. This is where IcEng get it's shader file. We need to call **IcEngJNI.copyAssetDir()** to copy shader file from asset to cache directory. Ref online for detail of Android asset management.

After init app, invoke the activity that contain IcEng View:

```
//---- Start MyTestActivity
Intent intent =
    new Intent(MainActivity.this, MyTestActivity.class);
MainActivity.this.startActivity(intent);
```

This **MyTestActivity** is where we want to show OpenGL stuff. In order to do that we put IcEngView into it's layout file **activit_my_test.xml** , like below :

```
<com.simviu.IcEng.IcEngView
    android:id="@+id/id_ic_eng_view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

You can put it some where in your layout. Refer online how to design your view by layout XML file. Once **MyTestActivity** is inflated, the IcEngView will start to render your scene based on you derived **IcApp**.

…. more detail coming soon...

Project Setup

…. more detail coming soon...

Directory structure:

- IcEng : Root of I-Cube 3d Engine
  - inc : include header path
  - src : Engine source code
  - doc : documents
  - demo : IcEng Demo src
  - proj : Project file for each platform
  - IcEngBuild : Binary build of Engine, for each platform
  - IcShader : default Shader

We provide project file for each platform.

- Mac : Xcode project
- iOS : Xcode project
- Linux : Use Cmake, by following commands:

  > cd ./IcEng/proj/linux/IcEng

> mkdir build

> cd build/

> cmake ..

> make

- Android : use Android studio to open project
- windows : Visual C++ project

License

Check BSD license :

https://en.wikipedia.org/wiki/BSD_licenses

Contact

support@simviu.com

…. more detail coming soon...