Developing an Interactive Graphics Application

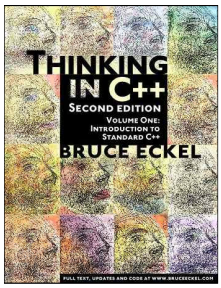
Computer Graphics Labs, Part II
Carlos Martinho, 2007 (translated 2009 by J. Brisson)

Outline

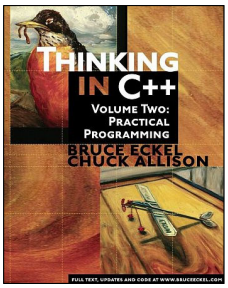
- Tools and Bibliography
- GCLib Tutorial

Tools Bibliography

Object Oriented Programming in C++

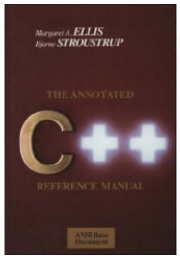


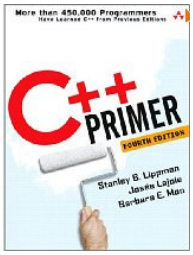
Free download



Free download

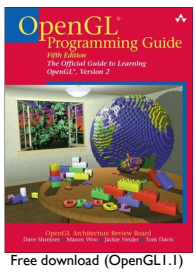
Object Oriented Programming in C++





OpenGL

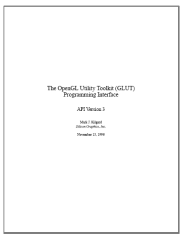
Software interface to graphics hardware.



Free download (OpenGL1.1)

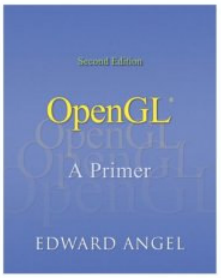
GLUT

OpenGL Toolkit Library
A window system independent toolkit for writing OpenGL programs.



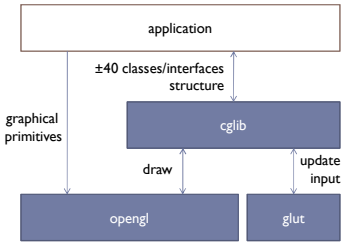
Free download version 3.7.6

OpenGL Primer



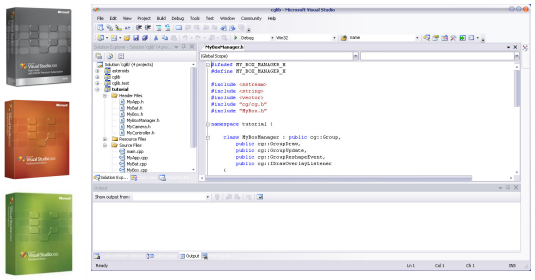
GPL Cglib

Object oriented library on top of OpenGL and GLUT



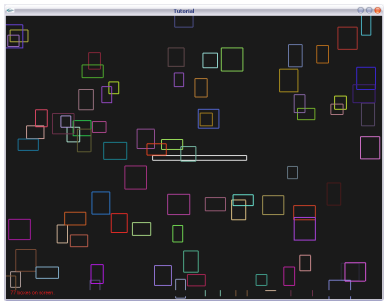
IDE: MS Visual Studio 2005

Native c++



Tutorial *cglib*

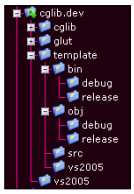
Objective



Step 0

Creating na integrated project with *cglib* inVS2005

Step 0: create a new VS2005 project



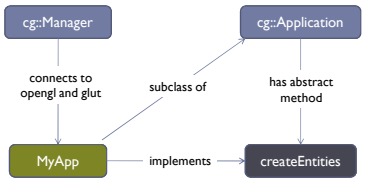
Workspace (solution) vs. Project
Project structure
Files required

Unpack workspace cglib.dev
Copy the "template" project
Change project name (see "readme.txt")
Add the project to the solution

Step 1

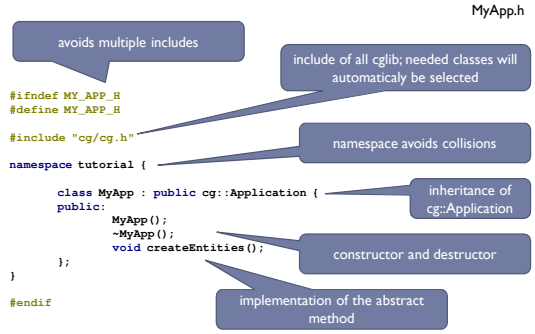
Creating an empty application

Step 1: create a cg::Application



Step 1: MyApp

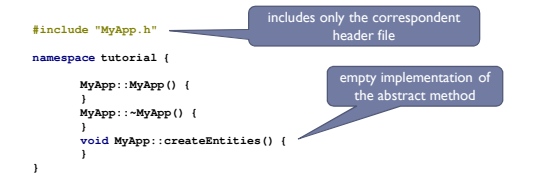
MyApp.h




```
#ifndef MY_APP_H
#define MY_APP_H

#include "cg/cg.h"

namespace tutorial {

	class MyApp : public cg::Application {
	public:
		MyApp();
		~MyApp();
		void createEntities();
	};
}

#endif
```

Step 1: MyApp

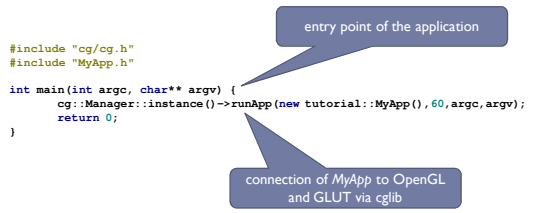
MyApp.cpp




```
#include "MyApp.h"

namespace tutorial {

	MyApp::MyApp() {
	}
	MyApp::~MyApp() {
	}
	void MyApp::createEntities() {
	}
}
```

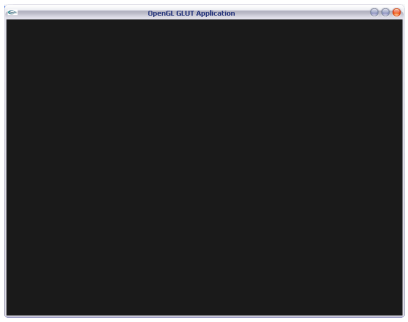
Step 1: main

main.cpp

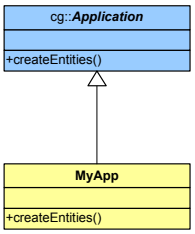



```
#include "cg/cg.h"
#include "MyApp.h"

int main(int argc, char** argv) {
	cg::Manager::instance()->runApp(new tutorial::MyApp(),60,argc,argv);
	return 0;
}
```

Step 1: running …



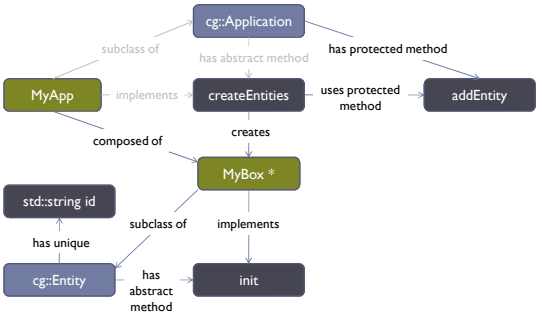
Step 1: summing up



Step 2

Creating a "box"

Step 2: create a cg::Entity



Step 2: MyApp::createEntities

MyApp.cpp

Loading a configuration file

```
#include "MyApp.h"

namespace tutorial {

        MyApp::MyApp() : cg::Application("config.ini") {
        }
        MyApp::~MyApp() {
        }
        void MyApp::createEntities() {
                addEntity(new MyBox("Box1"));
        }
}
```

Creating a new *MyBox* entity

Step 2: cg::Properties and cg::DebugFile

config.ini

```
MIN_SIZE = 20.0
MAX_SIZE = 50.0
```

log.txt

```
debug_text
more_debug_text
```

Properties [Singleton]

+load(in filename : string &)
+exists(in name : string &) : bool
+getInt(in name : string &) : int
+getFloat(in name : string &) : float
+getDouble(in name : string &) : double
+getString(in name : string &) : string
+getVector2d(in name : string &) : Vector2d
+getVector2i(in name : string &) : Vector2i
+getVector3d(in name : string &) : Vector3d
+getVector3i(in name : string &) : Vector3i

DebugFile [Singleton]

+getOutputFileStream() : ofstream &
+write(in s : string &)
+writeLine(in s : string &)
+newLine()
+writeException(in e : runtime_error &)

Step 2: MyBox

MyBox.h

```
#ifndef MY_BOX_H
#define MY_BOX_H

#include <string>
#include "cg/cg.h"

namespace tutorial {

        class MyBox : public cg::Entity {
        private:
                cg::Vector2d _position, _size;
                cg::Vector3d _color;
                double _winWidth, _winHeight;

                double randomBetween(double min, double max);

        public:
                MyBox(std::string id);
                ~MyBox();
                void init();
        };
}
#endif
```

Inheritance from *cg::Entity*

class that supports vectorial calculus

implementation of the abstract method

Step 2: MyBox

MyBox.cpp

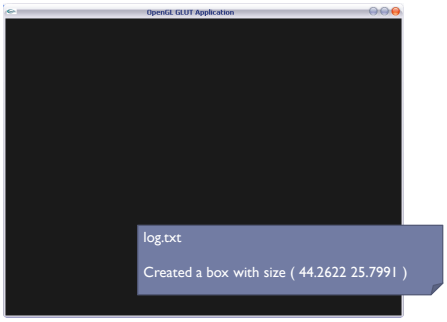
```
(...)
        MyBox::MyBox(std::string id) : cg::Entity(id) {
        }
        (...)
        void MyBox::init() {
                double min_size =
                        cg::Properties::instance()->getDouble("MIN_SIZE");
                (...)
                std::ofstream& file =
                        cg::DebugFile::instance()->getOutputFileStream();
                file << "Created a box with size " << _size << "." << std::endl;
        }
(...)
```

Passing a parameter to the parent class

Reading a parameter from the configuration file

Creation message written in the debug file

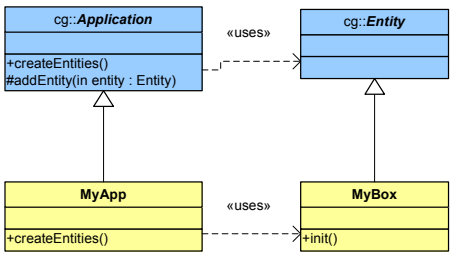
Step 2: ... running...



log.txt

Created a box with size (44.2622 25.7991)

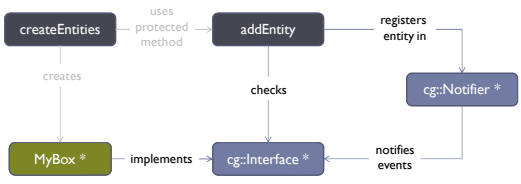
Step 2: summing up



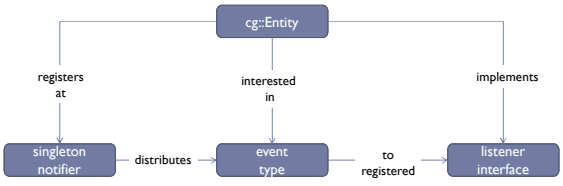
Step 3

Showing the "box" on the screen

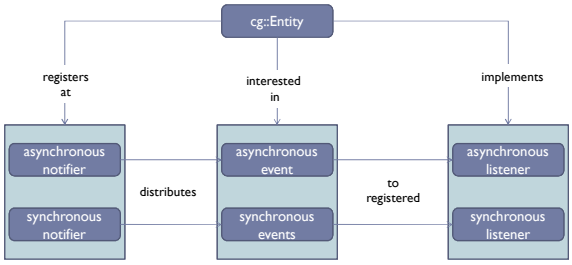
Step 3: notification interfaces



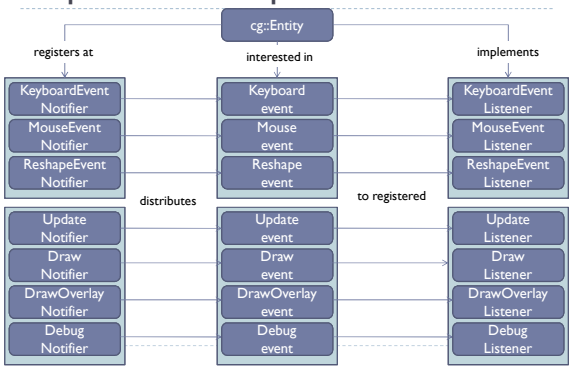
Step 3: notification pattern



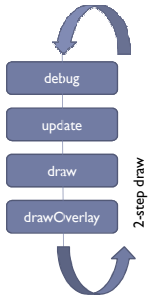
Step 3: notification pattern



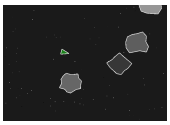
Step 3: notification pattern



Step 3: simulation cycle



draw



draw overlay

Step 3: MyBox

MyBox.h

```
(...)
        class MyBox : public cg::Entity, public cg::IDrawListener {
        private:
                (...)

        public:
                (...)
                void draw();
        };
}
(...)
```

implement *cg::IDrawListener*

Step 3

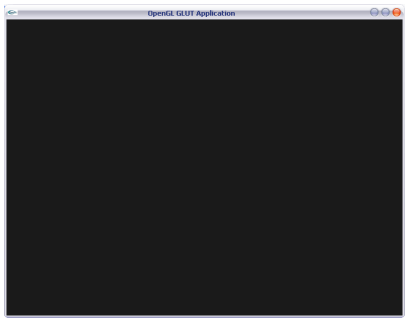
MyBox.cpp

Using cg::Vector

```
(...)
        void MyBox::draw() {
                cg::Vector2d min = _position - _size/2.0;
                cg::Vector2d max = _position + _size/2.0;
                glColor3dv(_color.get());
                glLineWidth(1.5);
                glBegin(GL_LINE_LOOP);
                        glVertex3d(min[0],min[1], 0);
                        glVertex3d(max[0],min[1], 0);
                        glVertex3d(max[0],max[1], 0);
                        glVertex3d(min[0],max[1], 0);
                glEnd();
        }
(...)
```

OpenGL

Step 3: a camera is missing



Step 3: MyCamera

MyCamera.h

```
(...)
    class MyCamera : public cg::Entity, public cg::IDrawListener {
    private:
        (...)
    public:
        MyCamera();
        virtual ~MyCamera();
        void init();
        void draw();
    };
(...)
```

The camera will define the projection in its draw method

Step 3: MyCamera

MyCamera.cpp

```
(...)
    MyCamera::MyCamera() : Entity("MyCamera") {}
    MyCamera::~MyCamera() {}
    void MyCamera::init() {
        (...)
    }
    void MyCamera::draw() {
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0,_winWidth,0,_winHeight,0,-100);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
    }
(...)
```

OpenGL code defining the orthogonal projection

Step 3: adding MyCamera to MyApp

MyApp.h

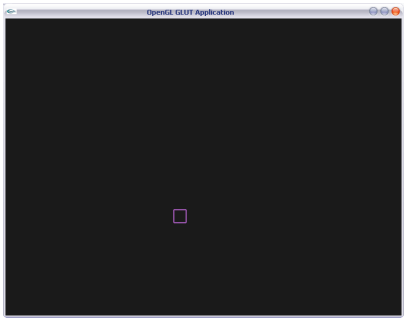
```
(...)
#include "MyCamera.h"
(...)
```

the camera is always the first cg::Entity

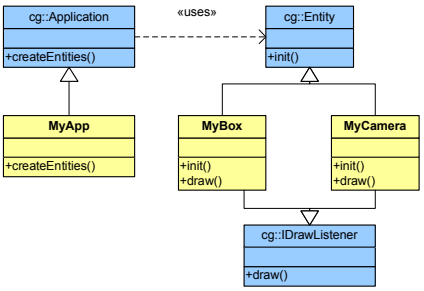
MyApp.cpp

```
(...)
        void MyApp::createEntities() {
                addEntity(new MyCamera());
                addEntity(new MyBox("Box1"));
        }
(...)
```

Step 3: ...running!



Step 3: summing up



Step 4

Animating the "box"

Step 4: MyBox

MyBox.h

```
(...)
class MyBox : public cg::Entity,
              public cg::IDrawListener,
              public cg::IUpdateListener
{
(...)
public:
        (...)
        void update(unsigned long elapsed_millis);
        (...)
};
(...)
```

implementing the *IUpdateListener* interface

Step 4: MyBox

reading from config.ini

creating the box

MyBox.cpp

```
(...)

void MyBox::init() {
    // Read from property file
    double min_size = cg::Properties::instance()->getDouble("MIN_SIZE");
    double max_size = cg::Properties::instance()->getDouble("MAX_SIZE");
    // Creates box
    cg::tWindow win = cg::Manager::instance()->getApp()->getWindow();
    _winWidth = win.width;
    _winHeight = win.height;
    _position = cg::Vector2d(randomBetween(0,_winWidth),randomBetween(0,_winHeight));
    _size = cg::Vector2d(randomBetween(min_size,max_size),randomBetween(min_size,max_size));
    _color = cg::Vector3d(randomBetween(0.1,0.9),randomBetween(0.1,0.9),randomBetween(0.1,0.9));
    _velocity = cg::Vector2d(randomBetween(100,300),randomBetween(100,300));
}
```

Step 4: MyBox

time to use in the simulation

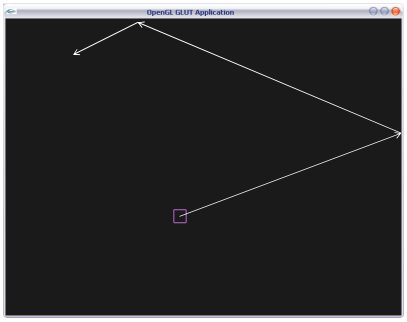
MyBox.cpp

```
void MyBox::update(unsigned long elapsed_millis) {
        double elapsed_seconds = elapsed_millis / 1000.0;
        _position += _velocity * elapsed_seconds;
        if(_position[0] < 0) {
                _position[0] = 0;
                _velocity[0] = -_velocity[0]; }
        if(_position[0] > _winWidth) {
                _position[0] = _winWidth;
                _velocity[0] = -_velocity[0]; }
        if(_position[1] < 0) {
                _position[1] = 0;
                _velocity[1] = -_velocity[1]; }
        if(_position[1] > _winHeight) {
                _position[1] = _winHeight;
                _velocity[1] = -_velocity[1]; }
}
(...)
```

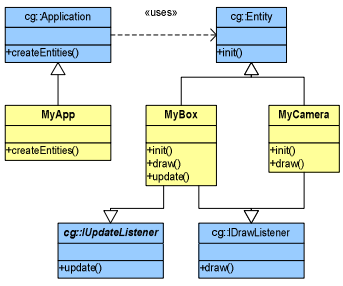
uniform velocity

collision with the screen limits (read in *init*)

Step 4: running!



Step 4: summing up



Step 5

Creating a mouse controlled "bat"

Step 5: MyBat

MyBat.h

```
class MyBat : public cg::Entity,
              public cg::IDrawListener,
              public cg::IMouseEventListener
{
        private:
                (...)
        public:
                (...)
                void init();
                void draw();
                void onMouse(int button, int state, int x, int y);
                void onMouseMotion(int x, int y);
                void onMousePassiveMotion(int x, int y);
};
```

implementing *IDrawListener* and *IMouseEventListener*

Bat behaviour responding to mouse events

Step 5: MyBat

MyBat.cpp

```
(...)
void MyBat::draw() {
        cg::Vector2d min = _position - _size/2.0;
        cg::Vector2d max = _position + _size/2.0;
        glColor3d(0.9,0.9,0.9);
        glLineWidth(1.5);
        glBegin(GL_LINE_LOOP);
                glVertex3d(min[0],min[1], 0);
                glVertex3d(max[0],min[1], 0);
                glVertex3d(max[0],max[1], 0);
                glVertex3d(min[0],max[1], 0);
        glEnd();
}
void MyBat::onMouse(int button, int state, int x, int y) {}
void MyBat::onMouseMotion(int x, int y) {}
void MyBat::onMousePassiveMotion(int x, int y) {
        _position[0] = x;
        _position[1] = _winHeight - y;
}
(...)
```

identical to *MyBox::draw* Sugests generalization

Object's position is the mouse's position

YY inversion

Step5: adding MyBat to MyApp

MyApp.h

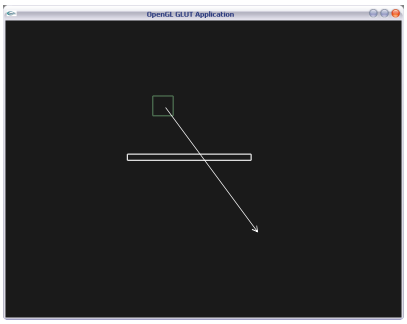
```
(...)
#include "MyBat.h"
(...)
```

MyApp.cpp

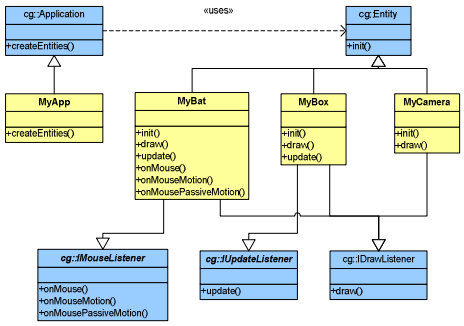
```
(...)
        void MyApp::createEntities() {
                addEntity(new MyCamera());
                addEntity(new MyBox("Box1"));
                addEntity(new MyBat("Bat"));
        }
(...)
```

new *cg::Entity*

Step 5: is running ...



Step 5: summing up



Step 6

Collisions of "bat" and "box"

Step 6: MyBox

MyBox.h

```
(...)
class MyBox : public cg::Entity,
              public cg::IDrawListener,
              public cg::IUpdateListener
{
private:
        (...)
        MyBat* _bat;
public:
        (...)
};
(...)
```

MyBox.cpp

```
(...)
void MyBox::init() {
        (...)
         _bat = (MyBat*)cg::Registry::instance()->get("Bat");
}
void MyBox::update(unsigned long elapsed_millis) {
        (...)
        if(_bat->isCollision(_position,_size)) {
                _velocity[1] = -_velocity[1]; }
}
(...)
```

get reference to the *bat* in *cg::Registry*

collision effect

Step 6: MyBat

MyBat.h

```
(...)
class MyBat : public cg::Entity,
              public cg::IDrawListener,
              public cg::IMouseEventListener {
(...)
public:
        (...)
        bool isCollision(cg::Vector2d box_position, cg::Vector2d box_size);
};
(...)
```

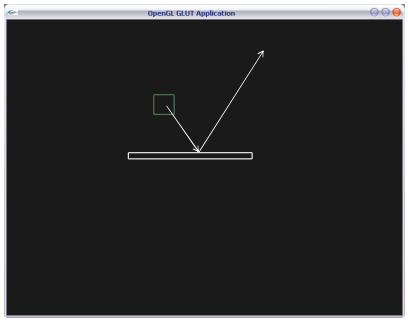
collision detection

use the collision method defined in *cg::Util*

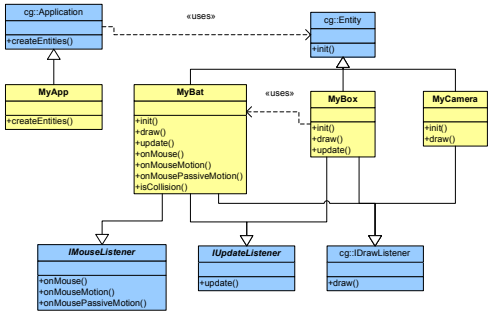
MyBat.cpp

```
(...)
bool MyBat::isCollision(cg::Vector2d box_position, cg::Vector2d box_size) {
        cg::Vector2d bat_bottomleft = _position - _size / 2.0;
        cg::Vector2d bat_topright = _position + _size / 2.0;
        cg::Vector2d box_bottomleft = box_position - box_size / 2.0;
        cg::Vector2d box_topright = box_position + box_size / 2.0;
        return cg::Util::instance()->isAABBoxCollision(
                bat_bottomleft,bat_topright,box_bottomleft,box_topright);
}
(...)
```

Step 6: is running!



Step 6: summing up



Step 7

Shall we create some more "boxes"?

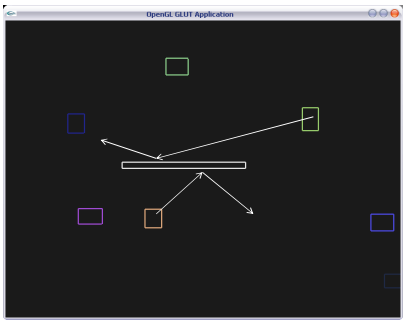
Step 7: MyApp

MyApp.cpp

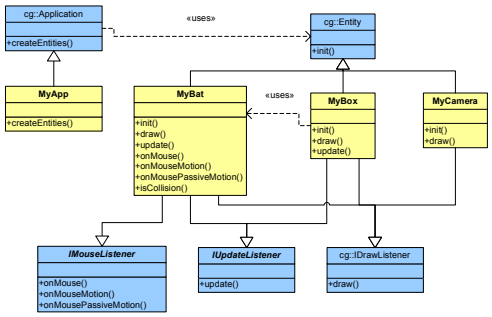
```
(...)
void MyApp::createEntities() {
        addEntity(new MyCamera());
        addEntity(new MyBox("Box1"));
        addEntity(new MyBox("Box2"));
        addEntity(new MyBox("Box3"));
        addEntity(new MyBox("Box4"));
        addEntity(new MyBox("Box5"));
        addEntity(new MyBox("Box6"));
        addEntity(new MyBox("Box7"));
        addEntity(new MyBat("Bat"));
}
(...)
```

not very extensible and inelegant

Step 7: yes, it's running!



Step 7: summing up (no changes)



Step 8

Creating any given number of "boxes"

Step 8: MyBoxManager

MyBoxManager.h

```
(...)
class MyBoxManager : public cg::Entity,
                     public cg::IDrawListener,
                     public cg::IUpdateListener,
                     public cg::IDrawOverlayListener {
private:
        std::vector<MyBox*> _boxes;
        typedef std::vector<MyBox*>::iterator tBoxIterator;
        std::string nboxMessage;

public:
        MyBoxManager(std::string id);
        ~MyBoxManager();
        void init();
        void update(unsigned long elapsed_millis);
        void draw();
        void drawOverlay();
};
(...)
```

using the STL (Standard Template Library)

new *IDrawOverlay* interface

Step 8: MyBoxManager

MyBoxManager.cpp

```
(...)
MyBoxManager::~MyBoxManager() {
        for(tBoxIterator i = _boxes.begin(); i != _boxes.end(); i++) {
                delete (*i);
        }
}

void MyBoxManager::init() {
        (...)
        int nbox = cg::Properties::instance()->getInt("NBOX");
        for(int i = 0; i < nbox; i++) {
                std::ostringstream os;
                os << "Box" << i;
                MyBox *box = new MyBox(os.str());
                box->init();
                _boxes.push_back(box);
        }
}
(...)
```

ecological programming

cg::Properties

dynamic creation and storage of *MyBox*

Step 8: MyBoxManager

MyBoxManager.cpp

```
(...)
void MyBoxManager::update(unsigned long elapsed_millis) {
        for(tBoxIterator i = _boxes.begin(); i != _boxes.end(); i++) {
                (*i)->update(elapsed_millis);
        }
}

void MyBoxManager::draw() {
        for(tBoxIterator i = _boxes.begin(); i != _boxes.end(); i++) {
                (*i)->draw();
        }
}

void MyBoxManager::drawOverlay() {
        glColor3d(0.9,0.1,0.1);
        cg::Util::instance()->drawBitmapString(nboxMessage,10,10);
}
(...)
```

event distribution

this event is not distributed

Step 8: adding MyBoxManager to MyApp

MyApp.h

```
(...)
#include "MyBoxManager.h"
(...)
```

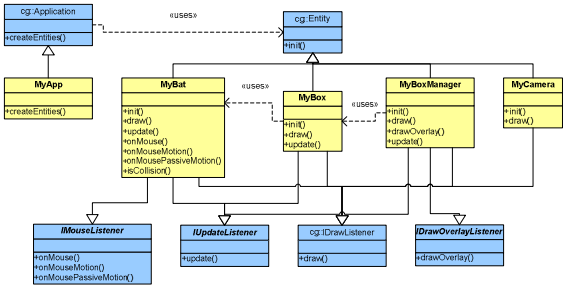
Replace the individual creation of the boxes

MyApp.cpp

```
(...)
        void MyApp::createEntities() {
                addEntity(new MyCamera());
                addEntity(new MyBoxManager("BoxManager"));
                addEntity(new MyBat("Bat"));
        }
(...)
```

Step 8: it's beautiful!

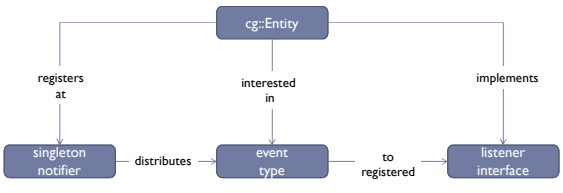
Step 8: summing up



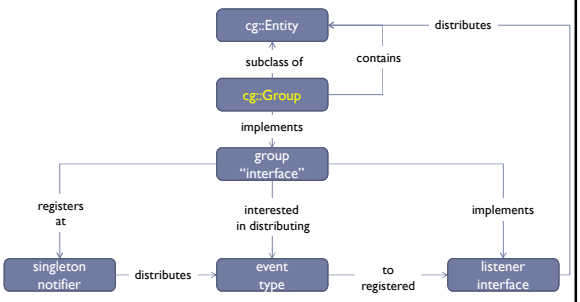
Step 9

Using cg::Group to manage enteties

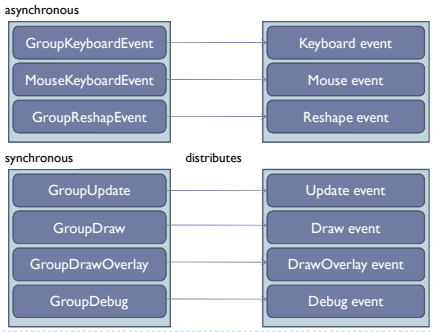
Step 9: notification pattern



Step 9: cg::Group



Step 9: group "interfaces"



Step 9: MyBoxManager

MyBoxManager.h

```
(...)
class MyBoxManager : public cg::Group,
                     public cg::GroupDraw,
                     public cg::GroupUpdate,
                     public cg::IDrawOverlayListener
{
(...)
protected:
        void createEntities();
        void postInit();
public:
        MyBoxManager(std::string id);
        ~MyBoxManager();
        void drawOverlay();
};
(...)
```

creation of *cg::Group*'s entities

method executed after the *cg::Entity*'s *init method*

Step 9: MyBoxManager

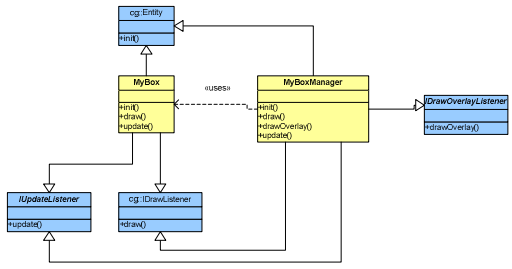
MyBoxManager.cpp

```
void MyBoxManager::createEntities() {
        int nbox = cg::Properties::instance()->getInt("NBOX");
        for(int i = 0; i < nbox; i++) {
                std::ostringstream os;
                os << "Box" << i;
                add(new MyBox(os.str()));
        }
}
void MyBoxManager::postInit() {
        std::ostringstream os;
        os << size() << " boxes on screen.";
        _nboxMessage = os.str();
}
void MyBoxManager::drawOverlay() {
        glColor3d(0.9,0.1,0.1);
        cg::Util::instance()->drawBitmapString(_nboxMessage,10,10);
}
```

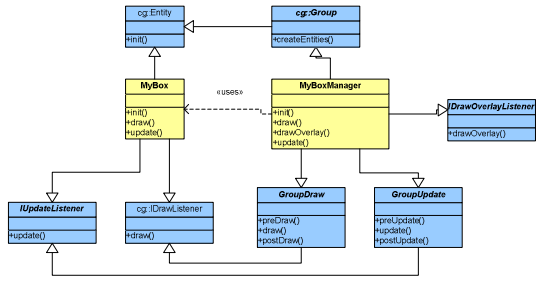
protected method in *cg::Group*

Step 9: (same?) result

Step 9: summing up (before)



Step 9: summing up (after)



Step 10

Application "shutdown" and "memory leaks"

Step 10: "memory leaks" in VS2005

main.cpp

```
//BEGIN: MEMORY LEAK DETECTION
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
//END: MEMORY LEAK DETECTION

(... includes ...)

int main(int argc, char** argv) {

//BEGIN: MEMORY LEAK DETECTION
       _CrtSetDbgFlag ( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
       //_CrtSetBreakAlloc(xxx);
//END: MEMORY LEAK DETECTION

        (... Código ...)
}
```

when the application ends, it refers which memory allocations have not been cleaned

allows to stop in the X memory allocation

Só está activo em modo debug.

Step 10: tutorial "memory leaks"!

```
Detected memory leaks!
Dumping objects ->
c:\program files\microsoft visual studio 8\vc\include\crtdbg.h(1147) : {1425}
normal block at 0x012EE9C0, 140 bytes long.
 Data: <d O  bQ  XQ   . > 64 F9 4F 00 80 62 51 10 0C 58 51 10 80 84 2E 01
(...)
c:\program files\microsoft visual studio 8\vc\include\crtdbg.h(1147) : {117}
normal block at 0x012E5FA0, 96 bytes long.
 Data: <h N             > 68 C7 4E 00 CD CD CD CD 00 00 00 00 CD CD CD CD
Object dump complete.
The program '[1112] tutorial.exe: Native' has exited with code 0 (0x0).
```

Step 10: "memory leaks" in VS2005

main.cpp

```
int main(int argc, char** argv) {

//BEGIN: MEMORY LEAK DETECTION
       _CrtSetDbgFlag ( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
       _CrtSetBreakAlloc(1363);
//END: MEMORY LEAK DETECTION

       (... Código ...)
}
```

stop at the 1363rd memory allocation

Step 10: MyController

MyController.h

```
(...)
class MyController : public cg::Entity,
                     public cg::IKeyboardEventListener
{
public:
       MyController();
       ~MyController();
       void init();
       void onKeyPressed(unsigned char key);
       void onKeyReleased(unsigned char key);
       void onSpecialKeyPressed(int key);
       void onSpecialKeyReleased(int key);
};
(...)
```

IKeyboardEventListener

Step 10: MyController

MyController.cpp

clean shutdown

```
void MyController::onKeyPressed(unsigned char key) {
        if (key == 27) {
                cg::Manager::instance()->shutdownApp();
        }
}
void MyController::onKeyReleased(unsigned char key) {
        if (key == ' ') {
                cg::Registry::instance()->get("BoxManager")->state.toggle();
        }
}
void MyController::onSpecialKeyPressed(int key) {
}
void MyController::onSpecialKeyReleased(int key) {
        if (key == GLUT_KEY_F1) {
                cg::Manager::instance()->getApp()->dump();
        }
}
```

changes the *BoxManager*'s state

GLUT special keys (arrow keys and F1 to F12)

structure dump to log.txt

Step 10: adding MyController to MyApp

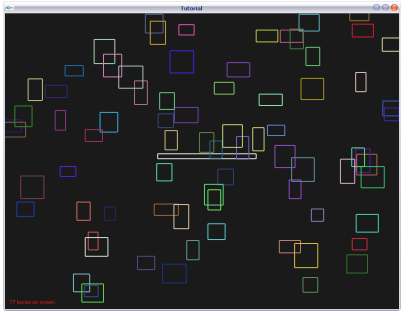
MyApp.h

```
(...)
#include "MyController.h"
(...)
```

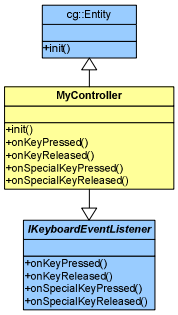
MyApp.cpp

```
(...)
        void MyApp::createEntities() {
                addEntity(new MyCamera());
                addEntity(new MyBoxManager("BoxManager"));
                addEntity(new MyBat("Bat"));
                addEntity(new MyController());
        }
(...)
```

Step 10: no more "memory leaks"



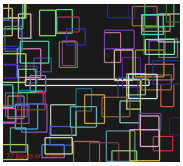
Step 10: summing up

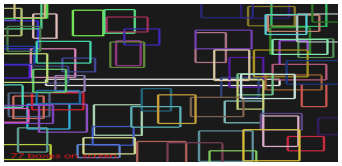


Step 11

Adapting to window resizing

Step 11: viewport deformations





Step 11: viewport deformations

MyCamera implementa IReshapeEventListener — MyCamera.h

```
(...)
class MyCamera : public cg::Entity,
                 public cg::IDrawListener,
                 public cg::IReshapeEventListener {
(...)
public:
        (...)
        void onReshape(int width, int height);
};
(...)
```

MyCamera.cpp

```
(...)
void MyCamera::draw() {
        (...)
        glOrtho(0,_winWidth,0,_winHeight,0,-100);
        (...)
}
void MyCamera::onReshape(int width, int height) {
        _winWidth = width;
        _winHeight = height;
}
(...)
```

Step 11: YY "mouse" coordinate

MyBat implementa IReshapeEventListener — MyBat.h

```
(...)
class MyBat : public cg::Entity,
              public cg::IDrawListener,
              public cg::IMouseEventListener,
              public cg::IReshapeEventListener
{
(...)
public:
        (...)
        void onReshape(int width, int height);
};
(...)
```

MyBat.cpp

```
(...)
void MyBat::onMousePassiveMotion(int x, int y) {
        _position[0] = x;
        _position[1] = _winHeight - y;
}
void MyBat::onReshape(int width, int height) {
            _winHeight = height;
        }
(...)
```

Step 11: colliding with window limits

MyBox implementa IReshapeEventListener — MyBox.h

```
(...)
class MyBox : public cg::Entity, public cg::IDrawListener,
              public cg::IUpdateListener,
              public cg::IReshapeEventListener  {
(...)
public:
        (...)
        void onReshape(int width, int height);
};
(...)
(...)
void MyBox::update(unsigned long elapsed_millis) {
        (...)
        if(_position[0] > _winWidth) { (... collide ...) }
        (...)
}
void MyBox::onReshape(int width, int height) {
        _winWidth = width;
        _winHeight = height;
}
(...)
```

MyBox.cpp

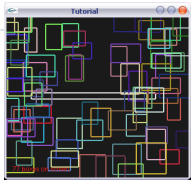
Step 11: initial window size

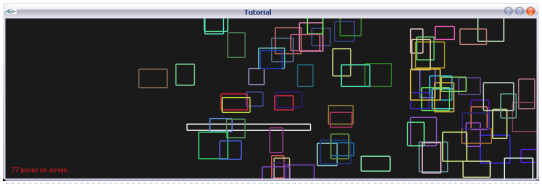
MyApp.cpp

```
(...)
MyApp::MyApp() : cg::Application("config.ini") {
        _window.caption = "Tutorial";
        _window.width = 800;
        _window.height = 600;
}
(...)
```

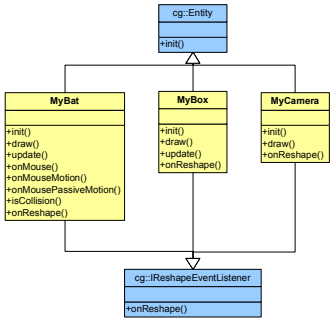
Step 11: running

Adapting to window space.





Step 11: summing up



The end (or the beginning?)

A beginning is a very delicate time...
Frank Herbert

To start:

- Install VS2005 (or later)
- Download c++, opengl and glut documentation
- Download cglib (zip) and tutorial (pdf and source)
- Follow the tutorial step by step
- Start working in your game...

Have a nice working day (and some fun!)