Contents

1 Introduction 1

2 Concepts 3

2.1 Ivf++ library structure . . . 3

2.2 Base class . . . 3

2.3 Smart pointers . . . 5

2.4 Renderable base class . . . 6

2.5 View management . . . 7

2.6 Materials . . . 9

2.7 Textures and images . . . 10

2.8 Visible classes . . . 11

2.9 Composite classes . . . 12

2.10 Lighting . . . 12

2.11 Controlling OpenGL state . . . 13

2.12 Rendering state information . . . 13

3 Ivf++ user interface library 15

3.1 A ivfui tutorial . . . 15

3.2 Using a scene object . . . 19

3.3 Responding to user input . . . 20

3.4 Idle processing . . . 23

3.5 Timer processing . . . 25

3.6 Assignable events . . . 30

3.7 Handler classes . . . 32

4 Using the scenegraph **33**

4.1 Placing and rotating objects 33

4.2 Creating a movable robot arm 35

4.3 Scene lighting 42

4.4 Textures 52

4.5 Fonts 58

5 Creating advanced geometry **59**

5.1 Advanced geometry 59

5.2 Extrusions 70

6 Rendering techniques **75**

6.1 Level of detail and switch objects 75

6.2 Scene graph culling 80

7 Advanced techniques **87**

7.1 Object selection 87

7.2 Loading 3d models 91

8 Using Ivf++ with the FLTK library **93**

9 Extending the Ivf++ library **103**

9.1 Derived composite classes 103

9.2 CIvfShape derived classes 105

9.3 CIvfGLPrimitive derived classes 107

9.4 Implementing type information 109

9.5 Declaring a smart pointers 110

10 Using Application and Class generators **111**

10.1 Ivf++ Application Generator 111

10.2 Ivf++ Class Generator 114

10.2.1 Generating classes from the command line 115

A Building and installing **121**

A.1 Building Ivf++ on Linux 121

A.1.1 Requirements 121

A.1.2 Unpacking the distribution 122
A.1.3 Downloading, building and installing dependencies 122
A.1.4 Building Ivf++ and installing 123
A.2 Building Ivf++ on Windows 124
A.2.1 Requirements 124
A.2.2 Unpacking the distribution 125
A.2.3 Configuring the Ivf++ build environment 127
A.2.4 Building Ivf++ 127
A.2.5 Visual C++ 2005 Express edition 129
A.2.6 Installing Ivf++ from the binary distribution 129

B Tutorial source code 131
B.1 Using Ivf++ user interface library 131
B.2 Placing and rotating objects 137
B.3 Creating a movable robot arm 142
B.4 Scene lighting 150
B.5 Textures 158
B.6 Advanced geometry 163
B.7 Extrusions 173
B.8 Level of detail and switch objects 179
B.9 Scene graph culling 186
B.10 Object selection 192
B.11 Loading 3d models 196

List of Figures

2.1 Ivf++ base classes 4

4.1 Object placement sample 36
4.2 Movable robot arm 36
4.3 Robot arm hierarchy 37
4.4 Finished robot arm 43
4.5 Spotlight cutoff angle 47
4.6 Exponent factor 48
4.7 Point light 51
4.8 Spotlight, Cutoff angle 70 degrees, Exponent 5 51
4.9 Spotlight, Cutoff angle 70 degrees, Exponent 30 52
4.10 Spotlight, Cutoff angle 25 degrees, Exponent 5 52
4.11 Texturing in Ivf++ 56
4.12 GL_NEAREST and GL_LINEAR filter settings 57
4.13 Texturing without mipmapping and with 57
4.14 GL_MODULATE and GL_BLEND modes with a red material 58

5.1 Index relationships 59
5.2 Resulting point set 63
5.3 Coordinate numbering 63
5.4 Lines using a linear index 65
5.5 *CIvfTriSet* object 67
5.6 Cube geometry 68
5.7 Colored cube using the CIvfQuadSet object 69
5.8 Spine geometry 70

5.9 *CIvfExtrusion* object 73

6.1 LOD transistion function 75
6.2 Objects with different LOD-shapes 80
6.3 Switch states 81
6.4 Culled scene from standard camera 85
6.5 Unculled and culled scene 86

7.1 Scene with sphere selected 90
7.2 AC3D model viewed in Ivf++ 92

8.1 Finished FLTK application with Ivf++ widget 93
8.2 FLUID user interface designer 94
8.3 New class dialog 94
8.4 FLUID with the new class CMainFrame 94
8.5 Adding a class constructor 95
8.6 Class constructor in the FLUID designer 95
8.7 The default window control 95
8.8 Window properties 96
8.9 FLUID window with window control 96
8.10 Properties for the ”box” control 97
8.11 Resized box-control 97
8.12 Callback code for the ”Idle” button 97
8.13 Finished application window 98
8.14 Code editor 98

9.1 *CIvfGLPrimitive* class structure 107

10.1 The Ivf++ Application Generator user interface 112
10.2 Application example parameters 114
10.3 Files generated by CMake 115

A.1 Unpacking the Ivf++ source package .gz-file 125
A.2 Unpacking the Ivf++ source package tar archive 125
A.3 Installing the Ivf++ dependency package 126
A.4 Ivf++ source and dependency directories ready for building 126

List of Tables

3.1 Handler classes in lvf++ 32

10.1 Handler classes in lvf++ 113

Preface

Chapter 1

Introduction

Ivf++ is a C++ library encapsulating OpenGL [1] functionality. The primary goal is to make it easier to use the OpenGL library [1] in interactive 3D applications. The second goal is extendibility, providing a set of well defined base classes for different object types to build new classes on. The last goal is portability, primarily between Linux and Windows, but the library should also be easily ported to Mac OS X.

In most of the chapters in this book the main concepts of the Ivf++ framework are described using a number of example programs. Most of these are based on the example program created in chapter 3.

Chapter 2

Concepts

It is often difficult to start using a class library. They often consist of a substantial number of classes. The classes can be used separately or in combination. Without knowing the concepts behind the library it can be difficult to use it. This chapter will describe the main concepts behind the Ivf++ library.

2.1 Ivf++ library structure

Most of the classes in the Ivf++ library are derived from three main abstract classes. CIvfBase handling type information and reference counting. This class is also the root class for all other classes in the library. CIvfObject handles all classes that use the OpenGL [1] library in any way. The CIvfShape class is used by all visual classes in the library. To make it possible to define a hierarchical scene graph the CIvfComposite class can contain a set of child objects derived from the CIvfShape class. CIvfTransform is used to create implement hierarchical models for example link robotic arms. The main classes are shown in figure 2.1.

2.2 Base class

The CIvfBase class is the common denominator of the class library. This class handles reference counting and type information. Reference counting in Ivf++ is used to enable sharing and simple destruction of instantiated classes. The basic idea is that when assigning an instantiated object to another the ownership of it is transferred to the latter. The following code shows how this works.

```
CIvfSphere* sphere = new CIvfSphere();
CIvfMaterial* material = new CIvfMaterial();

sphere->setMaterial(material);
```

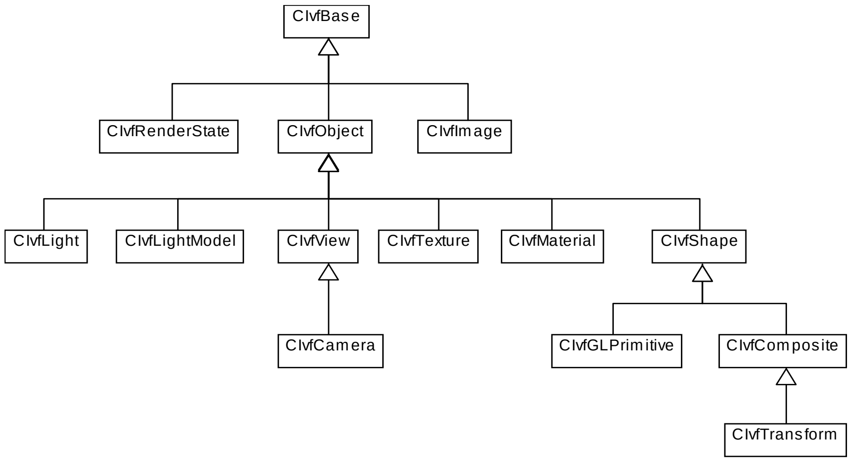


Figure 2.1: Ivf++ base classes

```
// More code here

delete sphere; // Sphere will delete material
```

The sphere object in the above code is responsible for destroying the material if no other object uses it. The code executed in the setMaterial method is shown below.

```
void CIvfShape::setMaterial (CIvfMaterial* material)
{
    if (m_material!=NULL)
    {
        m_material->deleteReference();
        if (!m_material->referenced())
            delete m_material;
    }
    m_material = material;
    m_material->addReference();
}
```

The m_material member variable is checked for an existing material. If one exists the reference count of this object is lowered (deleteReference), and then the existing material is queried to see if it is referenced. If not referenced (referenced) the previous material is removed. In the last step, the member variable m_material is assigned the new object and the reference count of this object is increased (addReference). To change the responsibility of the material destruction in the sphere example above, the code is modified as follows

```
CIvfSphere* sphere = new CIvfSphere();
CIvfMaterial* material = new CIvfMaterial();
material->addReference(); // We own the material now.

sphere->setMaterial(material);

// More code here

delete sphere;
delete material; // We must delete it
```

Using the addReference method on the material instance transfers the ownership back. It is then up to the user to delete the material.

Type information is implemented in a very simple way. Each object implements two methods, getClassName and isClass. The getClassName virtual method returns the name of the class. The isClass method returns true if the string passed to the function corresponds the class name of the current class or any of its inherited classes. The following code shows how this can be used to query shapes contained in a CIvfComposite object.

```
CIvfShape* shape;

for (i=0; i<composite->getSize(); i++)
{
    shape = composite->getChild(i);

    if (shape->isClass("CIvfCube"))
        cout << "CIvfCube object.";

    if (shape->isClass("CIvfSphere"))
        cout << "CIvfSphere object.";
}
```

2.3 Smart pointers

To make it easier to use reference counting Ivf++ defines smart pointers for most classes. A smart pointer is in itself a class which can be used like a pointer variable. The difference from a normal pointer variable is that it knows how Ivf++ reference counting system works. To use a smart pointer variable instead of a pointer variable, the * operator is replaced with Ptr. The following example illustrate the the use of smart pointers.

```
void mymethod()
{
    CIvfSpherePtr sphere = new CIvfSphere();
```

```
        // addReference() is called on sphere
    CIvfMaterialPtr material = new CIvfMaterial();
        // addReference() is called on material

    sphere->setMaterial(material);
        // addReference() is called on material
}
// deleteReference() is called on material by
    CIvfMaterialPtr.
// deleteReference() is called on sphere by
    CIvfMaterialPtr.
// deleteReference() is called on sphere by
    CIvfSpherePtr.
// sphere is deleted by CIvfSpherePtr
// material is deleted by CIvfSphere
```

All classes except the singleton classes have a associated smart pointer.

2.4 Renderable base class

Renderable classes in Ivf++ are derived directly from the CIvfObject class. Typical for these classes is that they can change the state of OpenGL when the render method is called. The are often used in some kind of response to a window refresh. There are two kinds of CIvfObject derived classes, visual and non-visual. Visual classes can be seen as some kind of rendered geometry in the OpenGL window. All visual classes are derived from the CIvfShape class. Non-visual classes are for example materials, textures and lights are derived directly from CIvfObject. Common to all CIvfObject classes is the render method, which calls the necessary functions to instantiate the object with the OpenGL library. CIvfObject also maintains object state using the setState method. The default state for all objects are set to CIvfObject::OS_ON. Setting the state to CIvfObject::OS_OFF will exclude the object from rendering. The class also maintains a select state, which is used to indicate an object selection. The default state of all objects are CIvfObject::SS_OFF. If the state is changed using the setSelect method to CIvfObject::SS_ON, special code will be called to render the selection geometry. To illustrate the workings of the render method it is shown below.

```
void CIvfObject::render ()
{
    if ((m_state == OS_ON)&&(!m_culled))
    {
        if (!m_useList)
        {
            if (m_renderState!=NULL)
                m_renderState->apply();

            beginTransform();
```

```
                if ((m_selectState == SS_ON)&&(
                    m_useSelectShape))
                     createSelect();

                createMaterial();
                createGeometry();
                endTransform();

                if (m_renderState!=NULL)
                     m_renderState->remove();
            }
            else
            {
                glCallList(getDisplayList());
            }
        }
        m_culled = false;
    }
```

The beginTransform, createSelect, createMaterial, createGeometry and endTransform are protected virtual members of the class and are implemented by the derived classes.

An effective technique for rendering in OpenGL is display lists. Display lists are recorded lists of OpenGL commands. Using display lists can have dramatic performance gains. The number of function calls to the OpenGL library is reduced. Many OpenGL implementations can often handle display lists very efficiently directly in hardware. Display lists in Ivf++ are used by setting the Uselist property to true with the setUselist method. Setting this property to true, any other changes to the objects will be frozen until the property is set to false. Note also that when setting the property to true on a composite object all objects contained will also be frozen. Before the setUselist method can be used a valid OpenGL graphics context must exist, which often means that the window for the OpenGL viewport must be visible.

2.5 View management

Perspective viewing in Ivf++ is implemented in the CIvfCamera class. This class uses the gluLookat function in the OpenGL utility library (GLU) to create the view transform. The class implements a number of methods for managing the camera. To use the camera a position and a target must be specified this is done using the setPosition and setTarget. The following code illustrates this.

```
    CIvfCameraPtr camera = new CIvfCamera();
    camera->setPosition(0.0, 5.0, 5.0);
    camera->setTarget(0.0, 0.0, 0.0);
```

The CIvfCamera class is typically used in conjunction with the onRender- and onResize-methods. In the onResize method the perspective transformation is adjusted to the viewport using the setViewport and initialize methods. In the onRender the camera transform is then applied using the render method of the camera. The following code shows the CIvfCamera class used with the onResize- and onRender-method of the Ivf++ user interface library (ivfui).

```
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_cube->render();
}

void CExampleWindow::onResize(int width, int height)
{
    m_camera->setPerspective(45.0, 0.1, 100.0);
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}
```

The onRender method of the camera object can seem a bit confusing at first. The concept of a camera does not exist in OpenGL. The viewer is always positioned on the Z-axis looking down the negative direction. The camera concept introduced with the *CIvfCamera* class simplifies the transformations and rotations needed to place the objects in a position that corresponds to a camera position and target. When the camera is "rendered" these transformations are applied. To be able to render objects correctly in a window the perspective transformation and viewport has to be defined.

The class also has a method pickVector, for calculating the vector going from a pixel in the viewport parallel with the view frustum. This vector can then be used to calculate intersections with other objects in the 3D space. Object selection is also supported by setting up a pick matrix using the initializeSelect method. The following code shows how the method is used to determine objects under a 4x4 pixel square on the screen.

```
glRenderMode(GL_SELECT);
glInitNames();
glPushName(-1);
glMatrixMode(GL_PROJECTION);
glPushMatrix();
    glLoadIdentity();
    m_camera->initializeSelect(x, y, 4, 4);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
        beginTransform();
        createGeometry();
        endTransform();
    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);

hits = glRenderMode(GL_RENDER);
```

It is also possible in Ivf++ to implement special view classes for creating custom view transforms. The CIvfCamera class itself is derived from the abstract base class CIvfView, which can be used to implement different types of view transforms.

2.6 Materials

Materials in the Ivf++ library are implemented in the CIvfMaterial class. The class simply maintains a set of member variables containing material properties. The material properties that can be set correspond to the properties that can be set by the glMaterial function in OpenGL. A typical usage of the CIvfMaterial class is shown below.

```
    CIvfMaterialPtr material = new CIvfMaterial();
    material->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    material->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f); . .
         .
    CIvfSpherePtr sphere = new CIvfSphere();
    sphere->setMaterial(material);
```

Materials are often assigned to CIvfShape derived classes and these classes render their materials in the createMaterial method.

```
    void CIvfShape::createMaterial()
    {
        if (CIvfGlobalState::getInstance()->
            isMaterialRenderingEnabled())
        {
            if (this->getSelect()==SS_ON)
            {
                if (m_selectMaterial!=NULL)
                    m_selectMaterial->render();
                else
                    if (m_material!=NULL)
                        m_material->render();
            }
            else
            {
                if (m_highlight == HS_ON)
                {
                    if (m_highlightMaterial!=NULL)
                        m_highlightMaterial->render();
                    else
                        if (m_material!=NULL)
                            m_material->render();
                }
                else
                    if (m_material!=NULL)
                        m_material->render();
            }
        }
    }
```

2.7 Textures and images

Textures are two-dimensional images that are mapped to 3D objects. Textures in the Ivf++ library are implemented in the CIvfTexture class. Using textures are done in the same way as a material; it often assigned to a CIvfShape derived class. Before a texture can be used an image has to be loaded. Currently the Ivf++ library supports images in the SGI image format (.rgb), JPEG format (.jpeg, .jpg) and PNG format(.png). The following code shows a typical use of the CIvfTexture class.

```
CIvfSgiImagePtr image = new CIvfSgiImage();
image->load("test.rgb");

CIvfTexturePtr texture = new CIvfTexture();
texture->setImage(image);

CIvfCubePtr cube = new CIvfCube();
cube->setTexture(texture);
```

Application of the texture is done in the CIvfShape beginTransform method. To conserve texture memory the texture is not loaded into memory before it is needed. The beginTransform method first tests to see if the texture is already in memory using the isBound method of the CIvfTexture class. If it is not loaded it is bound to memory using the bind method. Next time the beginTransform is called the texture is only selected using the apply method. The beginTransform code is shown below.

```
if (m_texture!=NULL)
{
    if (CIvfGlobalState::getInstance()->
        isTextureRenderingEnabled())
    {
        if (m_texture->isActive())
        {
                glPushAttrib(GL_TEXTURE_2D|
                    GL_TEXTURE_1D);
                glEnable(GL_TEXTURE_1D);
                glEnable(GL_TEXTURE_2D);
                if (!m_texture->isBound())
                    m_texture->bind();
                else
                    m_texture->apply();
            }
        }
    }
```

Textures can be applied to objects in different ways. Texture application is controlled by the setMode, setFilters and setGenerateMipmaps methods.

2.8 Visible classes

Visible classes are defined as classes that generate visible geometry in an OpenGL enabled window. In Ivf++, all visible classes are derived from the CIvfShape class. This section will describe some of the main CIvfShape derived classes. The CIvf-Shape class implements CIvfObjects virtual protected methods beginTransform, createMaterial, and endTransform with default implementations. This enables all CIvf-Shape derived classes to be rotated, positioned. Materials and textures can also be assigned. Positioning and rotation of objects can be done using the setPosition and setRotationQuat methods, which is illustrated in the following code.

```
CIvfCube* cube = new CIvfCube(); // Cube is a
    CIvfShape
cube->setPosition(1.0, 1.0, 1.0);
cube->setRotationQuat(1.0, 0.0, 0.0, 45.0);
```

Specifying a rotation using the setRotationQuat method is done by providing a rotation axis and a rotation angle. In the above code, the rotation axis is the X-axis and the rotation angle 45 degrees. The CIvfShape class also introduces the notion of highlighting objects. Highlighting objects can be done by calling the method setHighlight with either CIvfShape::HS_ON or CIvfShape::HS_OFF. When activated, the object will be drawn using the default highlight material, which is a white material. This material can also be changed using the setHighlightMaterial method. The CIvfShape class can also assign objects with a name that can be used to handle selection. Default all CIvfShape derived objects are assigned the name IVF_NONAME. Name usage is enabled and disabled by the setUseName method. The name of the object is set by the setObjectName method. The CIvfShape also provides a virtual refresh method that is used to update object geometry when this cannot be achieved in real-time.

2.9 Composite classes

To enable a hierarchical scene graphs the CIvfComposite object can be used contain other objects and other composite objects. In the tutorials, the CIvfComposite class is used as a scene object holding and rendering all objects in the scene. Calling the render method of the CIvfComposite class will loop through all child objects and call their render methods. Default, child objects added will be owned by the composite object, automatically destroying them when the composite object is destroyed. The following code shows how objects are added to a composite object.

```
CIvfSpherePtr sphere = new CIvfSphere();
CIvfCubePtr cube = new CIvfCube();

CIvfCompositePtr scene = new CIvfComposite();
scene->addChild(sphere);
scene->addChild(cube);

.
.
.
scene->render();
```

Positioning and rotating a composite object will rotate and move all child objects as well. A composite object can be seen as a user defined coordinate system that can be moved and rotated. Placing multiple composite objects inside each other can create hierarchical models, for example movable robot arms. To better represent this functionality, the composite derived class CIvfTransform can be used.

2.10 Lighting

Lighting in the Ivf++ library is implemented using the CIvfLighting singleton class. A singleton class is a class which there can be only one instance. The constructor of the CIvfLighting is protected and the instance of the class is retrieved using the method getInstance. The CIvfLight class handles the aspect of a single OpenGL light source. CIvfLighting handles a set of lights and OpenGL light state information. The following code shows how a point light source is created and used with the CIvfLighting singleton class.

```
m_lighting = CIvfLighting::getInstance();

CIvfLightPtr light = m_lighting->getLight(0);
light->setType(CIvfLight::LT_POINT);
light->setPosition(1.0, 1.0, 1.0);
light->setAmbientColor(0.2, 0.2, 0.2, 1.0);
```

Ivf++ supports directional light (CIvfLight::LT_DIRECTIONAL), point light source (CIvfLight::LT_POINT) and spot lights ((CIvfLight::LT_SPOT).

Rendering of the light is done when redrawing the window. In the Ivf++ user interface (ivfui) library, this is done in the onRender method, which is shown in the code below.

```
void CExampleWindow::onRender()
{
    m_lighting->render();
    m_camera->render();
    m_cube->render();
}
```

Placing the light before the camera object in the above code will have the effect of a headlight. Placing it after the camera will make it appear as a stationary light in the scene.

2.11 Controlling OpenGL state

OpenGL state is controlled using a set of singleton classes. Most common OpenGL states have been implemented. If some OpenGL state is not supported by Ivf++ it is no problem to call OpenGL directly.

- CIvfBlending for controlling OpenGL blending operations.
- CIvfFog for controlling OpenGL fog
- CIvfPixelOps for controlling for example OpenGL depth test
- CIvfRasterization controls rasterization operations in OpenGL

Retrieving instances of these classes can be done by calling the getInstance method. The following example shows how blending is enabled and the blend function defined.

```
CIvfBlendingPtr blending = CIvfBlending::getInstance()
    ;
blending->enable();
blending->defineAlphaBlendFunction();
```

2.12 Rendering state information

All Ivf++ classes derived from CIvfObject can be assigned a rendering state. This is done by assigning a CIvfRenderState derived class to the CIvfObject derived class using the setRenderState. Render states can be used to assign a specific render state to a node in the scene-graph. The Render state only affects the assigned node and its children. In the following example a special blend function is set for the sphere object.

```
CIvfSpherePtr sphere = new CIvfSphere();
CIvfBlendStatePtr blendState = new CIvfBlendState();
blendState->setFunction(GL_ONE, GL_ONE);
```

Render state is automatically restored when the object has finished rendering.

Chapter 3

Ivf++ user interface library

To create an OpenGL application a window with a suitable rendering context must be created. This process is quite different on the different platforms, making it hard to implement platform indpendent visualisation applications.

Drawing in OpenGL can also be complex. OpenGL can be seen as the assembly language of 3d graphics, knowledge of transformation matrices and linear algebra is necessary to fully utilise the library.

To make it easier to create an standard OpenGL application Ivf++, includes a special libraries to make it easier to create applications using OpenGL. Stand-alone OpenGL applications in Ivf++are implemented using the ivfui-library. Integration into other user interface libraries is done using the ivfwidget and its sublibraries ivffltk, ivfmfc and ivfwin32. These libraries encapsulates all the details and event handling needed to create a basic OpenGL application.

This chapter is focused mainly on the ivfui library, but most concepts applies equally when integrating Ivf++in other user interface toolkits using the ivffltk, ivfwin32 and ivfmfc libraries.

3.1 A ivfui tutorial

Implementing a simple OpenGL application using the ivfui-library consists of four steps:

- Adding the ivfui includes.
- Declaring a CIvfWindow derived window class.
- Declaring what events to be used.
- Implementing the main procedure.

This tutorial will not focus on the implementation details. For more detailed information see the tutorials in the following chapters.

In the first step the application window class is declared. The class will be derived from the CIvfWindow class. State variables for mouse handling is added and the needed Ivf++object declarations. The application will implemented in a single C++ source file. The class definition and necessary include directives are shown below.

```
#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfLighting.h>
#include <ivf/IvfCube.h>

class CExampleWindow: public CIvfWindow {
private:
    CIvfCameraPtr     m_camera;
    CIvfCubePtr       m_cube;
    CIvfLightPtr      m_light;
    CIvfLightingPtr   m_lighting;
public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
};
```

In the constructor declaration, the position and size of the window is specified using the X, Y, W, H parameters. In this example the constructor is left empty and the parameters passed directly to the parent class CIvfWindow. Initialisation in an *ivfui*-based application is done in the onInit method. This is to ensure that a valid OpenGL contexts exists before using any OpenGL calls are made. Finalisation of the window is done in the onDestroy method. To render and update the view transform the onRender and onResize methods are declared.

In this example a material, a cube, a light and a camera is created in the onInit method.

```
void CExampleWindow::onInit(int width, int height)
{
    // Initialize \ivf camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(2.0, 2.0, 2.0);
    m_camera->setPerspective(45.0, 0.1, 100.0);
```

```
        // Create a material

        CIvfMaterialPtr material = new CIvfMaterial();
        material->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
        material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f)
            ;
        material->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

        // Create a cube

        m_cube = new CIvfCube();
        m_cube->setMaterial(material);

        // Add lighting

        m_lighting = CIvfLighting::getInstance();

        m_light = m_lighting->getInstance();
        m_light->setPosition(1.0, 1.0, 1.0, 0.0);
        m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);

    }
```

To be able to make the window render something useful. The methods onRender and onResize also have to be implemented. The following code is added to the class declaration.

The onRender method is responsible for rendering the scene.

```
    void CExampleWindow::onRender()
    {
        m_lighting->render();
        m_camera->render();
        m_cube->render();
    }
```

The perspective transformation has to be modified if the window is reshaped. This is done in the onResize method.

```
    void CExampleWindow::onResize(int width, int height)
    {
        m_camera->setViewPort(width, height);
        m_camera->initialize();
    }
```

The application window class is now implemented and in the remaining part of the tutorial the main program of the application is created. First the application object is created:

```
int main(int argc, char **argv)
{
    // Create \ivf application object.

    CIvfApplicationPtr app = new CIvfApplication(
        IVF_DOUBLE|IVF_RGB);

    .
    .
```

Input to the CIvfApplication constructor is the desired visual of the application. In this case a double buffered rgb-display.

Next the window object is created.

```
    .
    .
    // Create a window

    CExampleWindowPtr window = new CExampleWindow(0,
        0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("\ivf␣Cube␣example");
    window->show();
    .
    .
```

The constructor for the CIvfWindow class takes the position and size of the window. In the example below the window is placed at (0,0) with the width and height set to 512. The window title is set using the setWindowTitle method. Finally the window is shown using the show method.

To enable the user to interact with the application, the event loop must be called using the run method of the CIvfApplication class.

```
    .
    .

    // Enter main application loop

    app->run();

    return 0;
}
```

When the user terminates the application the run method will exit and delete the application and window objects, and return.

A complete OpenGL application with which the user can interact with has now been created.

The complete source code of the example can be found in the end of this chapter.

3.2 Using a scene object

To simplify rendering and clean up, a composite object can be used to store all other objects in the scene. The scene object will call the render method of each object contained within it. This simplifies the program in that objects only have to be added in the onInit method. The onDestroy method is also simplified. The scene object will destroy any non-referenced object contained within it. To use a scene composite in this tutorial, a CIvfComposite object m_scene is added to the class definition.

```
          // \ivf object declarations

          CIvfCameraPtr      m_camera;
          CIvfCubePtr        m_cube;
          CIvfCompositePtr  m_scene; // Added
          CIvfLightPtr       m_light;
     public:
          CExampleWindow(int X, int Y, int W, int H)
               :CIvfWindow(X, Y, W, H) {};

          .
          .
```

In the onInit method the m_scene must created and objects used added to it using the addChild method.

```
     // Initialize scene

     m_scene = new CIvfComposite();

     // Create a cube

     m_cube = new CIvfCube();
     m_cube->setMaterial(material);

     m_scene->addChild(m_cube);
```

The onRender method is modified to render the m_scene object.

```
     void CExampleWindow::onRender()
     {
          m_lighting->render();
```

```
        m_camera->render();
        m_scene->render();
    }
```

3.3 Responding to user input

A *ivfui*based application can respond to user interaction from mouse and keyboard. The CIvfWindow class contains a number of virtual methods for responding to user input both from the keyboard and mouse. To make it possible to change the camera position with the mouse the following virtual methods are added to the class declaration in the previous example:

```
    class CExampleWindow: public CIvfWindow {
    private:

        // Camera movement state variables

        int m_beginX;
        int m_beginY;

        double m_angleX;
        double m_angleY;
        double m_moveX;
        double m_moveY;
        double m_zoomX;
        double m_zoomY;

        // \ivf object declarations

        CIvfCameraPtr      m_camera;
        CIvfCubePtr        m_cube;
        CIvfLightPtr       m_light;
        CIvfCompositePtr m_scene
    public:
        CExampleWindow(int X, int Y, int W, int H)
            :CIvfWindow(X, Y, W, H) {};

        // Basic ivfui event methods

        virtual void onInit(int width, int height);
        virtual void onResize(int width, int height);
        virtual void onRender();
        virtual void onDestroy();

        // Mouse event methods

        virtual void onMouseDown(int x, int y);
        virtual void onMouseMove(int x, int y);
        virtual void onMouseUp(int x, int y);
    };
```

State variables for handling camera movement are also added. To be able to respond to mouse movement, each mouse related event method receives as input the current mouse position. The camera will be rotated a certain amount based on the relative movement from the position were the mouse was pressed. In the onMouseDown method the initial mouse position will be stored in the m_beginX and m_beginY variables:

```
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}
```

In the onMouseMove method the relative distance from the initial position is calculated and transformed camera rotations, pan factors and zoom factors:

```
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (this->isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);

        this->redraw();
    }

    if (this->isRightButtonDown())
    {
        if (this->getModifierKey()==CIvfWidgetBase::
            MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
```

```
            else
            {
                m_moveX = (x - m_beginX);
                m_moveY = (y - m_beginY);
            }
            m_beginX = x;
            m_beginY = y;

            m_camera->moveSideways(m_moveX/100.0);
            m_camera->moveVertical(m_moveY/100.0);
            m_camera->moveDepth(m_zoomY/50.0);

            this->redraw();
        }
    }
```

The isLeftButtonDown and isRightButtonDown queries the current status of the mouse buttons. The getModifierKey method returns which additional keys are down on the keyboard. In the above example the camera can be zoomed by holding down the [shift] key and the right mouse button, while moving the mouse.

In the onMouseUp method the state variables for moving the camera are set to zero, so that the camera is not moved when the window is resized.

```
    void CExampleWindow::onMouseUp(int x, int y)
    {
        m_angleX = 0.0;
        m_angleY = 0.0;
        m_moveX = 0.0;
        m_moveY = 0.0;
        m_zoomX = 0.0;
        m_zoomY = 0.0;
    }
```

The CIvfCamera class has methods for moving and rotating the camera. This is added to the onRender method:

```
    void CExampleWindow::onRender()
    {
        m_lighting->render();
        m_camera->render();
        m_cube->render();
    }
```

The mouse can now be used to move the camera in different directions. Keyboard commands can also be used using the onKeyboard event method. In this example the 'r' key will be used to reset the camera position to the default position. First the onKeyboard method is added to the class definition:

```
        .
        .

        // Mouse event methods

        virtual void onMouseDown(int x, int y);
        virtual void onMouseMove(int x, int y);
        virtual void onMouseUp(int x, int y);

        // Keyboard event methods

        virtual void onKeyboard(int key, int x, int y);
    };
```

In the implementation of the onKeyboard event method, a switch statement is used to respond to different keys:

```
    void CExampleWindow::onKeyboard(int key, int x, int y)
    {
        switch (key) {
        case 'r':
            m_camera->setPosition(2.0, 2.0, 2.0);
            m_camera->setTarget(0.0, 0.0, 0.0);
            this->redraw();
            break;
        default:

            break;
        }
    }
```

In the above code the camera position and camera target are set to their default values and a "redraw" operation is initiated.

The complete source code of the example can be found in the end of this chapter.

3.4 Idle processing

The *ivfui*library supports idle processing. By idle processing is meant processing that occurs when the user is not interacting with the application. In *ivfui*idle processing can be implemented by adding a onIdle to the derived window class. Idle processing i the *ivfui*library is disabled by default. Enabling and disabling idle processing is done using the enableIdleProcessing and disableIdleProcessing methods. The state of idle processing can be queried using the method isIdleProcessing.

To illustrate the use of idle processing the previous example will be extended to rotate the camera around the cube, when idle processing is activated. First the onIdle is added to the class definition:

```
        .
        .

        // Keyboard event methods

        virtual void onKeyboard(int key, int x, int y);

        // Idle processing

        virtual void onIdle();
    };
```

In the implementation of the onIdle method the camera is rotated 0.1 degrees around the Y-axis and a call to redraw is done to redraw the window.

```
    void CExampleWindow::onIdle()
    {
        m_camera->rotatePositionY(0.1);
        this->redraw();
    }
```

This is a very simple implementation of an idle processing method. The rotation of the camera is highly dependent on the performance of the 3D graphics hardware. Ideally some kind of time measurement had to be made to rotate the camera at the same speed on any hardware.

If the example is compiled at this stage nothing will happen because idle processing is disabled by default in the *ivfui*library. To enable the idle processing we will add a key to the switch statement in the onKeyboard method to handle this:

```
    void CExampleWindow::onKeyboard(int key, int x, int y)
    {
        switch (key) {
        case 'r':
            m_camera->setPosition(2.0, 2.0, 2.0);
            m_camera->setTarget(0.0, 0.0, 0.0);
            this->redraw();
            break;
        case 'i':
            if (this->isIdleProcessing())
                this->disableIdleProcessing();
            else
                this->enableIdleProcessing();
            break;
        default:

            break;
        }
    }
```

It is important to understand that idle processing should be used with great care, because it can consumes a lot of CPU time when used. It is often better to user timer call backs, described in the next section, instead.

3.5 Timer processing

The *ivfui*library supports up to 10 independent timers and timer events. To illustrate the use of timers, the cube will be rotated 1 degree every 10:th of a second. Timer events are handled by the methods onTimeoutn, where n is a number from 0-9. To implement the cube rotation the onTimeout0 is added to the class implementation:

```
        .
        .

        // Idle processing

        virtual void onIdle();

        // Timer processing

        virtual bool onTimeout0();
    };
```

In the implementation of the method the cube rotation about the Y-axis is added 1.0 degrees. Then the redraw is called to initiate a redraw of the window.

```
    bool CExampleWindow::onTimeout0()
    {
        double rx, ry, rz, angle;

        m_cube->getRotationQuat(rx, ry, rz, angle);
        m_cube->setRotationQuat(1.0, 0.0, 0.0, angle+1.0);
        this->redraw();

        return true;
    }
```

Returning true in the onTimeout0 method will repeat the timer event.

Just adding a onTimeoutn method will not enable the timer event. To enable a timer event the enableTimeout method is used. The input parameters are a time interval and the number of the timeout event to be used. In the current example the timer event will be enabled in the onKeyboard method.

```
        .
```

```
            .

            case 't':
                if (this->isTimeoutEnabled(0))
                    this->disableTimeout(0);
                else
                    this->enableTimeout(0.1, 0);
                break;
            default:

                break;
            }
        }
```

When running this example, pressing the 't' key will enable the timer 0, which will start rotating the cube. Pressing 't' again will disable the timer callback onTimeout0.

The complete source code for the example in this chapter is shown in the following section.

Listing 3.1: Simple ivfui example

```
// ------------------------------------------------------------
//
// Ivf++ colored cube example
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfLighting.h>
#include <ivf/IvfCube.h>
#include <ivf/IvfComposite.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

IvfSmartPointer(CExampleWindow);

class CExampleWindow: public CIvfWindow {
private:

        // Camera movement state variables
        int m_beginX;
        int m_beginY;

        double m_angleX;
        double m_angleY;
        double m_moveX;
        double m_moveY;
        double m_zoomX;
        double m_zoomY;

        // Ivf++ object declarations

        CIvfCameraPtr           m_camera;
        CIvfCubePtr             m_cube;
        CIvfCompositePtr        m_scene;
        CIvfLightPtr            m_light;
public:
        CExampleWindow(int X, int Y, int W, int H)
                :CIvfWindow(X, Y, W, H) {};

        // Basic ivfui event methods

        virtual void onInit(int width, int height);
        virtual void onResize(int width, int height);
        virtual void onRender();

        // Mouse event methods

        virtual void onMouseDown(int x, int y);
        virtual void onMouseMove(int x, int y);
        virtual void onMouseUp(int x, int y);

        // Keyboard event methods

        virtual void onKeyboard(int key, int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
        // State variables

        m_angleX = 0.0f;
        m_angleY = 0.0f;
        m_moveX = 0.0f;
        m_moveY = 0.0f;
        m_zoomX = 0.0f;
        m_zoomY = 0.0f;

        // Initialize Ivf++ camera

        m_camera = new CIvfCamera();
        m_camera->setPosition(2.0, 2.0, 2.0);
        m_camera->setPerspective(45.0, 0.1, 100.0);

        // Create a material
        CIvfMaterialPtr material = new CIvfMaterial();
        material->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
        material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
        material->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

        // Initialize scene

        m_scene = new CIvfComposite();

        // Create a cube

        m_cube = new CIvfCube();
        m_cube->setMaterial(material);

        m_scene->addChild(m_cube);

        // Create a light

        CIvfLightingPtr lighting = CIvfLighting::getInstance();

        m_light = lighting->getLight(0);
        m_light->setLightPosition(1.0, 1.0, 1.0);
        m_light->setAmbientColor(0.2f, 0.2f, 0.2f, 1.0f);
        m_light->enable();
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
        m_camera->setViewPort(width, height);
        m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
        m_light->render();
        m_camera->render();
        m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
        m_beginX = x;
        m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
    if (this->isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

                m_camera->rotatePositionY(m_angleX/100.0);
                m_camera->rotatePositionX(m_angleY/100.0);

                this->redraw();
    }

    if (this->isRightButtonDown())
    {
                if (this->getModifierKey()==CIvfWidgetBase::MT_SHIFT)
                {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }

        m_beginX = x;
        m_beginY = y;

                m_camera->moveSideways(m_moveX/100.0);
                m_camera->moveVertical(m_moveY/100.0);
                m_camera->moveDepth(m_zoomY/50.0);

        this->redraw();
    }


}

// ----------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
        m_angleX = 0.0;
        m_angleY = 0.0;
        m_moveX = 0.0;
        m_moveY = 0.0;
        m_zoomX = 0.0;
        m_zoomY = 0.0;
}

// ----------------------------------------------------------------
void CExampleWindow::onKeyboard(int key, int x, int y)
{
        switch (key) {
        case 'r':
                m_camera->setPosition(2.0, 2.0, 2.0);
                m_camera->setTarget(0.0, 0.0, 0.0);
                this->redraw();
                break;
        default:

                break;
        }
}

// ------------------------------------------------------------------------
// Main program
// ------------------------------------------------------------------------

int main(int argc, char **argv)
{
        // Create Ivf++ application object.

        CIvfApplicationPtr app = new CIvfApplication(IVF_DOUBLE|
          IVF_RGB);

        // Create a window

        CExampleWindowPtr window = new CExampleWindow(0, 0, 512, 512);

        // Set window title and show window

        window->setWindowTitle("Ivf++ Cube example");
        window->show();

        // Enter main application loop

        app->run();

        return 0;
}
```

Listing 3.1: Simple ivfui example

3.6 Assignable events

In the latest version of Ivf++a new event model has been introduced. In the previous event model virtual event methods are defined in the base class and then implemented in the child class. This model is not very flexible and can not support modular functionality. To solve this a new event model has been introduced. In this model all events are declared as separate classes with a single virtual method representing the event method. Because the events are classes they can easily be assigned to instance variables or vectors enabling assignment of multiple endpoints to an event. The new event model also enables the addition of event methods to any class, using multiple inheritance.

To implement the previous example using assignable events the needed events are added in the class inheritance declaration using multiple inheritance.

```
class CExampleWindow: public CIvfWindow,
    CIvfInitEvent,
        CIvfResizeEvent,
        CIvfRenderEvent,
        CIvfMouseDownEvent,
        CIvfMouseMoveEvent,
        CIvfMouseUpEvent,
        CIvfKeyboardEvent
{
private:
.
.
.
```

The existing virtual methods can be kept because the methods declared in the event classes are the same as using the old event model. The old and new event models can also be combined. The existing virtual event method of CIvfWidgetBase derived classes will be called in addition to the assigned event class instances.

Next the events must be assigned to using the methods in the CIvfWidgetBase class. This is best done in the constructor of the derived class.

```
class CExampleWindow: public CIvfWindow,
.
.
.
public:
        CExampleWindow(int X, int Y, int W, int H);

        .
        .
        .

};

CExampleWindow::CExampleWindow(int X, int Y, int W,
        int H)
        :CIvfWindow(X, Y, W, H)
{
        addInitEvent(this);
        addResizeEvent(this);
        addRenderEvent(this);
        addMouseDownEvent(this);
}
```

The example can now be recompiled and should work as in the previous version of the. The real benefit of the new event model is that multiple event methods can be assigned to a single event handled by the base class, enabling pluggable functionality using special ”handler” classes.

Handler class	description
CIvfSceneHandler	Handles rendering of a scene with resizable window.
CIvfMouseViewHandler	Handles moving and rotating a camera using the mouse.
CIvfSelectionHandler	Handles the object selection.
CIvfCoordinateInputHandler	Handles 3D coordinate input
CIvfFlyHandler	Handles moving and rotating a camera using a fly model.
CIvfInteractionHandler	Handles interaction with ivf3dui derived objects
CIvfShapePlacementHandler	Handles placement of shapes in a scene (BETA)

Table 3.1: Handler classes in Ivf++

3.7 Handler classes

Taking advantage of the assignable events as described in section 3.6, can be done by using some of the handler classes provided by Ivf++. Using these classes makes it even easier to create an interactive visualisation application. Table 3.1 lists some of the handler classes in Ivf++.

Chapter 4

Using the scenegraph

This chapters describes techniques for managing and rendering a Ivf++ scene graph. All topics are described in a tutorial form using based on the example program described in chapter 3.

4.1 Placing and rotating objects

This tutorial will extend the previous tutorial with additional objects. These new objects will be positioned and rotated to illustrate object placement and rotation methods. The new objects will also be given new materials. First some additional include directives must be added to the example.

```
#include <ivf/IvfCamera.h>
#include <ivf/IvfCube.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCylinder.h>
#include <ivf/IvfCone.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
```

The camera position is changed to (0.0, 4.0, 9.0) in the onInit method, to be able to see all objects in the scene.

```
// Initialize Ivf++ camera

m_camera = new CIvfCamera();
m_camera->setPosition(0.0, 4.0, 9.0);
```

Next some materials are created to be used by our objects. Note that the pointers to the materials can be declared locally, because they will be assigned to and destroyed by the scene objects.

```
// Create a materials

CIvfMaterialPtr redMaterial = new CIvfMaterial();
redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
redMaterial->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

CIvfMaterialPtr greenMaterial = new CIvfMaterial();
greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f)
    ;
greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f
    );
greenMaterial->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f)
    ;

CIvfMaterialPtr blueMaterial = new CIvfMaterial();
blueMaterial->setDiffuseColor(0.0f, 0.0f, 1.0f, 1.0f);
blueMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f)
    ;
blueMaterial->setAmbientColor(0.0f, 0.0f, 0.5f, 1.0f);

CIvfMaterialPtr yellowMaterial = new CIvfMaterial();
yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f
    );
yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0
    f);
yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f
    );
```

A red cube will be created and placed at (2.0, 0.0, 2.0). The cube will be assigned the redMaterial material, using the setMaterial method. Positioning the object is done using the setPosition method. All classes derived from CIvfShape will have this method. The cube will also be added to the scene object. This is done using the addChild method of the composite object.

```
// Create scene composite

m_scene = new CIvfComposite();

// Create objects

CIvfCubePtr cube = new CIvfCube();
cube->setMaterial(redMaterial);
cube->setPosition(2.0, 0.0, 2.0);
m_scene->addChild(cube);
```

The other objects in the scene will be created in the same way.

```
CIvfSpherePtr sphere = new CIvfSphere();
sphere->setMaterial(greenMaterial);
sphere->setPosition(-2.0, 0.0, 2.0);
m_scene->addChild(sphere);

CIvfCylinderPtr cylinder = new CIvfCylinder();
cylinder->setMaterial(blueMaterial);
cylinder->setPosition(-2.0, 0.0, -2.0);
m_scene->addChild(cylinder);

CIvfConePtr cone = new CIvfCone();
cone->setMaterial(yellowMaterial);
cone->setPosition(2.0, 0.0, -2.0);
cone->setRotationQuat(0.0, 0.0, 1.0, 45.0);
m_scene->addChild(cone);

CIvfAxisPtr axis = new CIvfAxis();
m_scene->addChild(axis);
```

The last object added axis is an axis indicator that displays the positive coordinate directions X, Y and Z as red, green and blue arrows.

All CIvfShape derived classes can be rotated using the setRotationQuat method. Using this method, the cone will be given a 45-degree rotation around the Z-axis. The "right-hand" rule is used when rotating objects.

```
cone = new CIvfCone();
cone->setMaterial(yellowMaterial);
cone->setPosition(-2.0, 0.0, -2.0);
cone->setRotationQuat(0.0, 0.0, 1.0, 45.0);
```

The resulting program window is shown in figure 4.1.

The source code of the program is shown in appendix B.2

4.2 Creating a movable robot arm

To illustrate some of the more advanced modelling techniques in Ivf++, this tutorial will build a movable robot arm with sizes and objects as in figure 4.2.

To create the arm, hierarchical transforms will be applied using CIvfTransform objects. The structure of the objects is seen in Figure 4.3.

Below is shown the needed includes and modified class definition.

```
#include <ivf/IvfCamera.h>
```

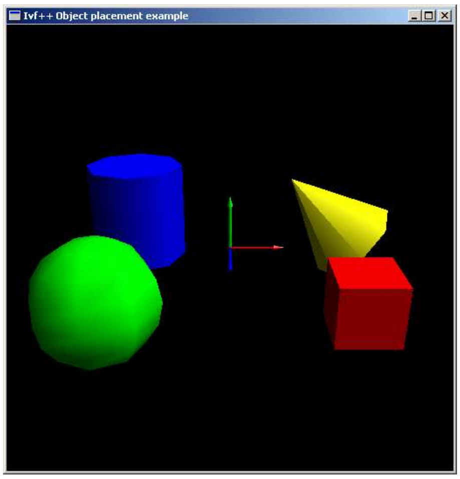


Figure 4.1: Object placement sample

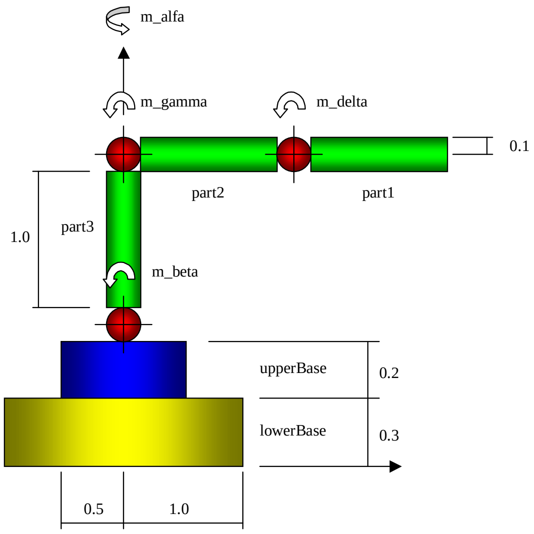


Figure 4.2: Movable robot arm

```
#include <ivf/IvfSphere.h>
#include <ivf/IvfCylinder.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLighting.h>
#include <ivf/IvfMaterial.h>

// ------------------------------------------------------------
```

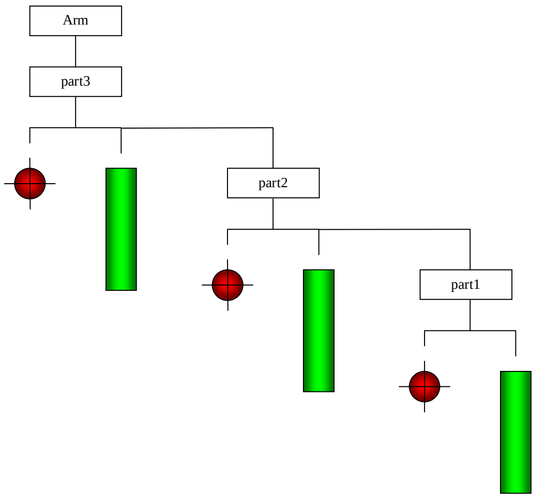


Figure 4.3: Robot arm hierarchy

```
// Window class definition
// ---------------------------------------------------------------

class CExampleWindow : public CIvfWindow {
private:

    // Camera movement state variables

    int m_beginX;
    int m_beginY;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    CIvfCameraPtr      m_camera;
    CIvfLightingPtr    m_lighting;
    CIvfLightPtr       m_light;
    CIvfCompositePtr   m_scene;

    // Robot state variables

    double m_alfa;
    double m_beta;
    double m_gamma;
```

```
        double m_delta;

        // Robot arm transforms

        CIvfTransformPtr  m_part1;
        CIvfTransformPtr  m_part2;
        CIvfTransformPtr  m_part3;
        CIvfTransformPtr  m_arm;

        // Routine for updating the arm

        void updateArm();
    public:
        CExampleWindow(int X, int Y, int W, int H)
            :CIvfWindow(X, Y, W, H) {};
        .
        .
```

The variables m_alfa, m_beta, m_gamma and m_delta are used to define the different rotations of the robot arm. The CIvfTransform objects are used to create the hierarchical model of the robot arm. The updateArm is used to update the rotations of the robot arm.

In the onInit method we add the initialisation of the robot arm angle variables:

```
    m_alfa = 0.0;
    m_beta = -45.0;
    m_gamma = 90.0;
    m_delta = 75.0;
```

The camera is placed at (0.0, 5.0, 5.0)

```
    m_camera->setPosition(0.0, 5.0, 5.0);
```

The same material definitions and scene object as in section 4.1 are used.

```
    // Create a materials

    CIvfMaterialPtr redMaterial = new CIvfMaterial();
    redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    redMaterial->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

    CIvfMaterialPtr greenMaterial = new CIvfMaterial();
    greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f)
        ;
    greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f
        );
```

```
greenMaterial->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f)
    ;

CIvfMaterialPtr blueMaterial = new CIvfMaterial();
blueMaterial->setDiffuseColor(0.0f, 0.0f, 1.0f, 1.0f);
blueMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f)
    ;
blueMaterial->setAmbientColor(0.0f, 0.0f, 0.5f, 1.0f);

CIvfMaterialPtr yellowMaterial = new CIvfMaterial();
yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f
    );
yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0
    f);
yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f
    );

// Create scene composite

m_scene = new CIvfComposite();
```

After the scene object creation, the robot arm can be created. The first object created on the robot is the base, on which the arm is placed. Two cylinders are used to create the base.

```
// Create robot base

CIvfCylinderPtr lowerBase = new CIvfCylinder();
lowerBase->setHeight(0.2);
lowerBase->setMaterial(yellowMaterial);
m_scene->addChild(lowerBase);

CIvfCylinderPtr upperBase = new CIvfCylinder();
upperBase->setHeight(0.3);
upperBase->setRadius(0.5);
upperBase->setPosition(0.0, 0.1 + 0.15, 0.0);
upperBase->setMaterial(blueMaterial);
m_scene->addChild(upperBase);
```

The arm is built in reverse order starting from part1 to part3 and finally the complete arm object. Each part of the arm is placed in a CIvfTransform object. A transform object can be described as a user coordinate system. By placing each movable part of the arm in a transform object it is possible to link them together to form a hierarchical transform chain. Each transform object in the chain will transform all of its children. As an example the part2 transform object will transform all children including the part1 transform object, this in turn will transform its children. The following code shows how the different parts of the robot arm are assembled with Ivf++ objects. Note that each part of the arm is created using a local coordinate system starting with the sphere at (0.0, 0.0, 0.0) and the cylinder placed on top. This places the rotational centre of each part in the centre of the spheres. Each part is the placed on top of each other. part1 is placed at (0.0, 1.2, 0.0) relative the part2 coordinate system.

```
// Create part1

m_part1 = new CIvfTransform();

CIvfSpherePtr sphere = new CIvfSphere();
sphere->setRadius(0.1);
sphere->setMaterial(redMaterial);
m_part1->addChild(sphere);

CIvfCylinderPtr cylinder = new CIvfCylinder();
cylinder->setRadius(0.1);
cylinder->setHeight(1.0);
cylinder->setMaterial(greenMaterial);
cylinder->setPosition(0.0, 0.1 + 0.5, 0.0);

m_part1->addChild(cylinder);
m_part1->setPosition(0.0, 1.2, 0.0);
m_part1->setRotationQuat(0.0, 0.0, 1.0, m_delta);

// Create part2

m_part2 = new CIvfTransform();

sphere = new CIvfSphere();
sphere->setRadius(0.1);
sphere->setMaterial(redMaterial);
m_part2->addChild(sphere);

cylinder = new CIvfCylinder();
cylinder->setRadius(0.1);
cylinder->setHeight(1.0);
cylinder->setMaterial(greenMaterial);
cylinder->setPosition(0.0, 0.1 + 0.5, 0.0);
m_part2->addChild(cylinder);
m_part2->addChild(m_part1);
m_part2->setPosition(0.0, 1.2, 0.0);
m_part2->setRotationQuat(0.0, 0.0, 1.0, m_gamma);

// Create part3

m_part3 = new CIvfTransform();

sphere = new CIvfSphere();
sphere->setRadius(0.1);
sphere->setMaterial(redMaterial);
m_part3->addChild(sphere);
```

```
cylinder = new CIvfCylinder();
cylinder->setRadius(0.1);
cylinder->setHeight(1.0);
cylinder->setMaterial(greenMaterial);
cylinder->setPosition(0.0, 0.1 + 0.5, 0.0);
m_part3->addChild(cylinder);
m_part3->addChild(m_part2);
m_part3->setPosition(0.0, 0.0, 0.0);
m_part3->setRotationQuat(0.0, 0.0, 1.0, m_beta);
```

To be able to rotate the entire arm around the Y-axis, all parts are placed in the arm transform. See also figure 4.3.

```
// Create complete arm

m_arm = new CIvfTransform();
m_arm->addChild(m_part3);
m_arm->setPosition(0.0, 0.1+0.4, 0.0);
m_arm->setRotationQuat(0.0, 1.0, 0.0, m_alfa);

m_scene->addChild(m_arm);
```

To enable easy updating of the robot arm a updateArm routine is created to apply the arm rotation and tell the GLUT library to refresh the window.

```
void CExampleWindow::updateArm()
{
    m_arm->setRotationQuat(0.0, 1.0, 0.0, m_alfa);
    m_part1->setRotationQuat(0.0, 0.0, 1.0, m_delta);
    m_part2->setRotationQuat(0.0, 0.0, 1.0, m_gamma);
    m_part3->setRotationQuat(0.0, 0.0, 1.0, m_beta);

    redraw();
}
```

To enable the user to manipulate the arm, the onKeyboard is modified. Additional cases are added to the switch statement. In each of the new cases the *updateArm* routine is called to update the arm and redraw the window.

```
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    switch (key) {
    case 'a':
        m_alfa += 5.0;
        updateArm();
        break;
    case 'z':
        m_alfa -= 5.0;
        updateArm();
        break;
    case 's':
        m_beta += 5.0;
        updateArm();
        break;
    case 'x':
        m_beta -= 5.0;
        updateArm();
        break;
    case 'd':
        m_gamma += 5.0;
        updateArm();
        break;
    case 'c':
        m_gamma -= 5.0;
        updateArm();
        break;
    case 'f':
        m_delta += 5.0;
        updateArm();
        break;
    case 'v':
        m_delta -= 5.0;
        updateArm();
        break;
    default:
        break;
    }
}
```

The resulting application window is shown in figure 4.4.

The source code of the program is shown in the next section.

4.3 Scene lighting

Lighting in Ivf++ is implemented in the CIvfLight and CIvfLightModel classes. The CIvfLight implements a single light source and CIvfLightModel handles a set of lights. To illustrate the use of lighting in a scene, an example of a light moving in a room built of 6 CIvfMesh objects will be used.

First the necessary includes are defined.

```
#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
```

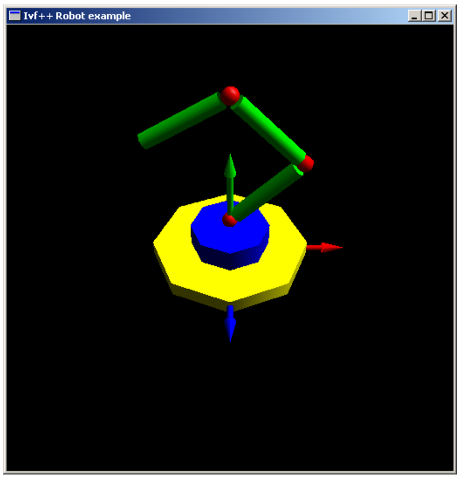


Figure 4.4: Finished robot arm

```
#include <ivf/IvfSphere.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfLighting.h>
#include <ivf/IvfMesh.h>
#include <ivf/IvfLightingState.h>
#include <ivf/IvfRasterization.h>

#include <ivfmath/IvfVec3d.h>
```

In the class definition member variables for the light and light model are added (m_light, m_lightModel. A composite object (m_scene) is used to manage the scene. Variables for moving the light are also added (m_direction, m_speed.

```
        .
        .
        CIvfCameraPtr          m_camera;
        CIvfCompositePtr       m_scene;
        CIvfLightPtr           m_light;
        CIvfLightingPtr        m_lighting;
        CIvfSpherePtr          m_lightSphere;

        CIvfVec3d              m_direction;
        double                 m_speed;
public:
        CExampleWindow(int X, int Y, int W, int H)
                :CIvfWindow(X, Y, W, H) {};

        .
        .
```

To implement the movement of the light a timeout method onTimeout0 is added to the class definition.

```
        .
        .

        // Keyboard events

        virtual void onKeyboard(int key, int x, int y);

        // Timer events

        virtual bool onTimeout0();
    };
```

In the onInit method the camera position set to (0.0, 0.0, 9.0) and the field of view is adjusted to 60 degrees.

```
    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 0.0, 9.0);
    m_camera->setPerspective(60.0, 0.1, 40.0);
```

A set of material and a scene composite are created.

```
    // Create a materials

    CIvfMaterialPtr redMaterial = new CIvfMaterial();
    redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    redMaterial->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

    CIvfMaterialPtr greenMaterial = new CIvfMaterial();
    greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f)
        ;
    greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f
        );
    greenMaterial->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f)
        ;

    CIvfMaterialPtr blueMaterial = new CIvfMaterial();
    blueMaterial->setDiffuseColor(0.0f, 0.0f, 1.0f, 1.0f);
    blueMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f)
        ;
    blueMaterial->setAmbientColor(0.0f, 0.0f, 0.5f, 1.0f);

    CIvfMaterialPtr yellowMaterial = new CIvfMaterial();
    yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f
        );
    yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0
        f);
    yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f
        );

    CIvfMaterialPtr whiteMaterial = new CIvfMaterial();
    whiteMaterial->setDiffuseColor(1.0f, 1.0f, 1.0f, 1.0f)
        ;
    whiteMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f
        );
    whiteMaterial->setAmbientColor(0.5f, 0.5f, 0.5f, 1.0f)
        ;

    // Create scene composite

    m_scene = new CIvfComposite();
```

To illustrate the light position a small sphere object m_lightSphere is created. Because the sphere will be located at the same position as the light it will not be lighted correctly. To solve this lighting will be switched off when rendering the sphere. To do this a CIvfRenderState derived object CIvfLightingState will be assigned to the sphere. This object encapsulates an OpenGL state change. In the render method, the state change will be applied before any geometry is rendered, and removed at the end of the rendering process:

```
// Create a sphere representing the light

CIvfLightingStatePtr lightState = new
    CIvfLightingState();
lightState->setLighting(false);

m_lightSphere = new CIvfSphere();
m_lightSphere->setMaterial(whiteMaterial);
m_lightSphere->setRadius(0.1);
m_lightSphere->setRenderState(lightState);
m_lightSphere->setPosition(-2.0, 0.0, 2.0);
m_scene->addChild(m_lightSphere);
```

A larger red sphere is created in the middle of the room to illustrate the lighting effects on a smooth object.

```
// Create a sphere in the middle

CIvfSpherePtr sphere = new CIvfSphere();
sphere->setRadius(1.0);
sphere->setMaterial(redMaterial);
sphere->setStacks(12);
```

```
    sphere->setSlices(20);
    m_scene->addChild(sphere);
```

Lighting in OpenGL is calculated on a per-surface basis. To be able to get a good picture of the lighting effects the walls of the rooms are created using CIvfMesh objects using a 30x30 mesh. Because the walls are flat the walls are created using a second order mesh (MT_ORDER_2).

```
    // Create a room

    CIvfMeshPtr floor = new CIvfMesh();
    floor->setMeshType(CIvfMesh::MT_ORDER_2);
    floor->setMeshResolution(30,30);
    floor->createMesh(10.0, 10.0);
    floor->setMaterial(redMaterial);
    floor->setPosition(0.0, -3.0, 0.0);

    m_scene->addChild(floor);

    CIvfMeshPtr roof = new CIvfMesh();
    roof->setMeshType(CIvfMesh::MT_ORDER_2);
    roof->setMeshResolution(30,30);
    roof->createMesh(10.0, 10.0);
    roof->setMaterial(greenMaterial);
    roof->setPosition(0.0, 3.0, 0.0);
    roof->setRotationQuat(0.0, 0.0, 1.0, 180.0f);
    m_scene->addChild(roof);

    CIvfMeshPtr wall1 = new CIvfMesh();
    wall1->setMeshType(CIvfMesh::MT_ORDER_2);
    wall1->setMeshResolution(30,30);
    wall1->createMesh(10.0, 6.0);
    wall1->setMaterial(blueMaterial);
    wall1->setPosition(0.0, 0.0, -5.0);
    wall1->setRotationQuat(1.0, 0.0, 0.0, 90.0f);
    m_scene->addChild(wall1);

    CIvfMeshPtr wall2 = new CIvfMesh();
    wall2->setMeshType(CIvfMesh::MT_ORDER_2);
    wall2->setMeshResolution(30,30);
    wall2->createMesh(10.0, 6.0);
    wall2->setMaterial(blueMaterial);
    wall2->setPosition(0.0, 0.0, 5.0);
    wall2->setRotationQuat(1.0, 0.0, 0.0, -90.0f);
    m_scene->addChild(wall2);

    CIvfMeshPtr wall3 = new CIvfMesh();
    wall3->setMeshType(CIvfMesh::MT_ORDER_2);
    wall3->setMeshResolution(30,30);
    wall3->createMesh(6.0, 10.0);
```

```
wall3->setMaterial(yellowMaterial);
wall3->setPosition(5.0, 0.0, 0.0);
wall3->setRotationQuat(0.0, 0.0, 1.0, 90.0f);
m_scene->addChild(wall3);

CIvfMeshPtr wall4 = new CIvfMesh();
wall4->setMeshType(CIvfMesh::MT_ORDER_2);
wall4->setMeshResolution(30,30);
wall4->createMesh(6.0, 10.0);
wall4->setMaterial(yellowMaterial);
wall4->setPosition(-5.0, 0.0, 0.0);
wall4->setRotationQuat(0.0, 0.0, 1.0, -90.0f);
m_scene->addChild(wall4);
```

In the next step the light and light model will be setup. The spotlight is controlled by the light direction, cutoff angle and the falloff exponent. The cutoff angle determines the size of the spotlight cone. The exponent factor determines how focused the spotlight is. Figures 4.5 and 4.6 illustrates these parameters.

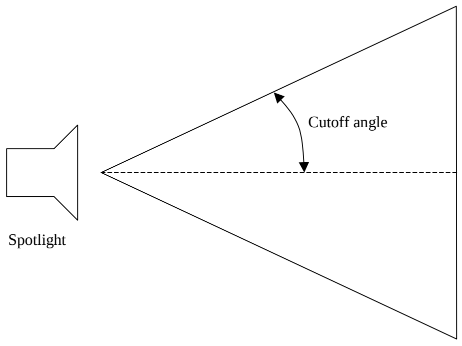


Figure 4.5: Spotlight cutoff angle

The initial position is set to (-2.0, 0.0, 2.0) and the diffuse and ambient colors are set to white and dark gray. Finally the newly created light is added to the light model.

```
// Create a light

m_lighting = CIvfLighting::getInstance();

m_light = m_lighting->getLight(0);
m_light->setType(CIvfLight::LT_POINT);
m_light->setLightPosition(-2.0, 0.0, 2.0);
m_light->setSpotCutoff(70.0f);
m_light->setSpotExponent(20.0f);
m_light->setDiffuseColor(1.0f, 1.0f, 1.0f, 1.0f);
m_light->setAmbientColor(0.2f, 0.2f, 0.2f, 1.0f);
m_light->enable();
```

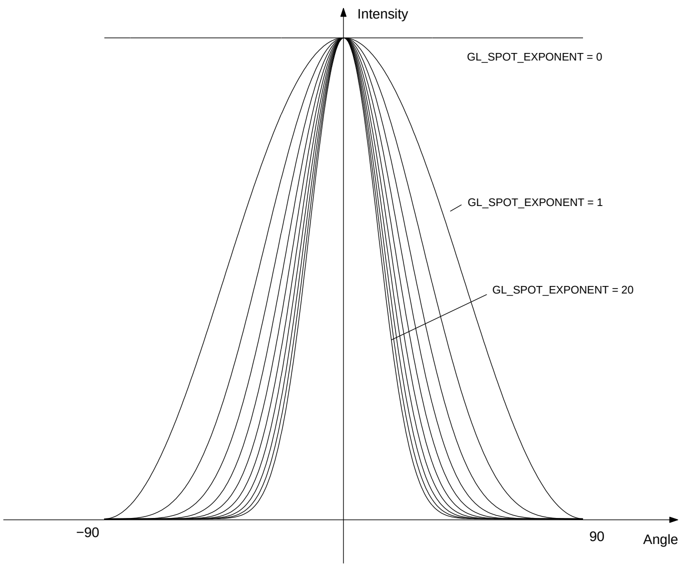


Figure 4.6: Exponent factor

To be able to zoom out and still be able to see the inside of the room, back face culling is switched on.

```
CIvfRasterizationPtr rasterOps = CIvfRasterization::
    getInstance();
rasterOps->enableCullFace();
rasterOps->setCullFace(CIvfRasterization::CF_BACK);
```

In the final part of the onInit method the moving light is initialised and the onTimeout0 method enabled.

```
// Setup moving light

m_direction.setComponents(1.0, 1.0, 1.0);
m_speed = 0.06;

enableTimeout(0.01, 0);
```

The onRender method must be modified to render the scene composite m_scene and light model m_lightmodel.

```
void CExampleWindow::onRender()
{
    m_camera->render();
    m_lighting->render();
```

```
        m_scene->render();
    }
```

Here the light model is rendered after the camera, which means that the lights will placed in the scene coordinate system.

In the onKeyboard method the switch method will be extended with keys for changing the different parameters of the light object. First a key 'l' will be added to be able to change the light type from point source to spotlight.

```
    void CExampleWindow::onKeyboard(int key, int x, int y)
    {
        float angle;
        float exponent;

        switch (key) {
        case 'l':
            if (m_light->getType()==CIvfLight::LT_POINT)
                m_light->setType(CIvfLight::LT_SPOT);
            else
                m_light->setType(CIvfLight::LT_POINT);
            break;
        .
        .
```

The spotlight cutoff angle is controlled by the setSpotCutoff method and the 'a' and 'z' keys will be used to modify this parameter in steps of 5 degrees.

```
    case 'a':
        angle = m_light->getSpotCutoff();
        angle += 5.0f;
        cout << "Angle = " << angle  << endl;
        m_light->setSpotCutoff(angle);
        break;
    case 'z':
        angle = m_light->getSpotCutoff();
        angle -= 5.0f;
        cout << "Angle = " << angle  << endl;
        m_light->setSpotCutoff(angle);
        break;
```

The spotlight exponent factor is controlled by the setSpotExponent method and the 'a' and 'z' keys will be used to modify this parameter in steps of 1.

```
    case 's':
        exponent = m_light->getSpotExponent();
        exponent += 1.0f;
        cout << "Exponent = " << exponent  << endl;
        m_light->setSpotExponent(exponent);
        break;
    case 'x':
        exponent = m_light->getSpotExponent();
        exponent -= 1.0f;
        cout << "Exponent = " << exponent << endl;
        m_light->setSpotExponent(exponent);
        break;
```

Finally the speed of the light is changed using the 'd' and 'c' keys.

```
    case 'd':
        m_speed += 0.01;
        break;
    case 'c':
        m_speed -= 0.01;
        break;
    default:
        break;
```

Movement of the sphere and light is controller in the onTimeout0 method. The idea is that the light will bounce of the walls in the room. When the light is a spotlight, the direction of the spotlight will be in the direction of the movement. The onTimeout0 returns true, so that it will be repeated continously.

```
    bool CExampleWindow::onTimeout0()
    {
        CIvfVec3d pos;
        double x, y, z;
        double ex, ey, ez;

        pos = m_lightSphere->getPosition();
        pos = pos + m_direction * m_speed;

        pos.getComponents(x, y, z);
        m_direction.getComponents(ex, ey, ez);

        if ((x>4.8)||(x<-4.8))
            ex = -ex;

        if ((y>2.8)||(y<-2.8))
            ey = -ey;

        if ((z>4.8)||(z<-4.8))
            ez = -ez;

        m_direction.setComponents(ex, ey, ez);

        m_lightSphere->setPosition(pos);
        pos.getComponents(x, y, z);
        m_light->setPosition(x, y, z);
        m_light->setSpotDirection(ex, ey, ez);

        this->redraw();

        return true;
    }
```

The results of running the program are shown in figures 4.7, 4.8, 4.9 and 4.10

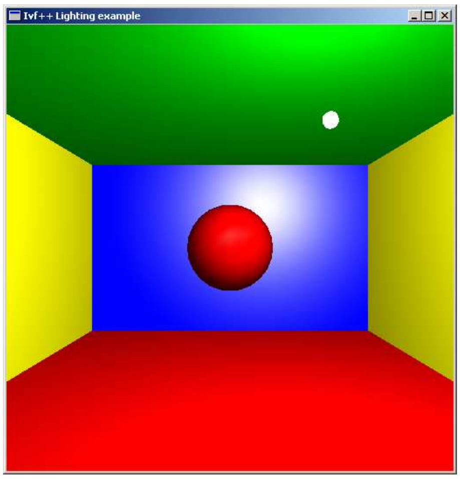


Figure 4.7: Point light

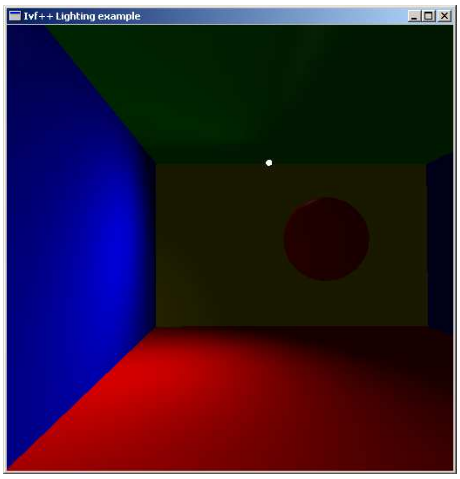


Figure 4.8: Spotlight, Cutoff angle 70 degrees, Exponent 5

The source code of the program is shown in the next section.

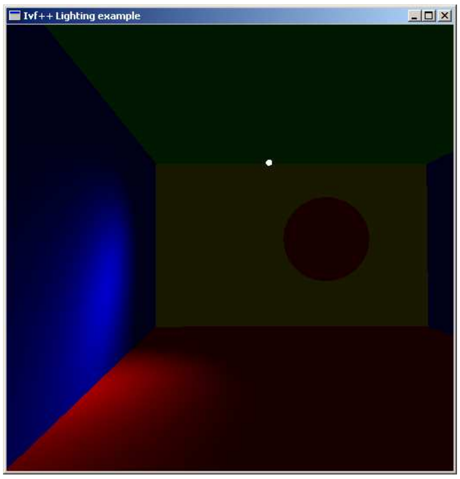


Figure 4.9: Spotlight, Cutoff angle 70 degrees, Exponent 30

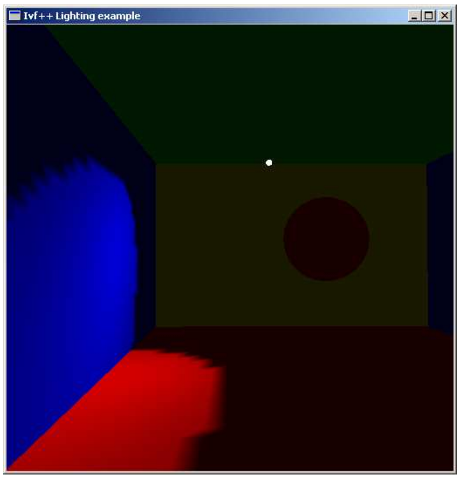


Figure 4.10: Spotlight, Cutoff angle 25 degrees, Exponent 5

4.4 Textures

Textures in Ivf++ are applied to objects in the same way as materials. To be able to show a texture on an object in Ivf++, the object has to render texture coordinates. Most of Ivf++ objects are able to render texture coordinates automatically. In this example an image will be loaded and placed on a CIvfQuadPlane. Keyboard commands will be used to illustrate the different texture settings available.

The following code shows the necessary includes.

```
#include <ivfui/IvfApplication.h>
```

```
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfTexture.h>
#include <ivf/IvfQuadPlane.h>

#include <ivfimage/IvfPngImage.h>
```

In the class definition a member variable for the texture is added so that it can be accessed from the onKeyboard method. A member for the scene composite m_scene is added. The event method onKeyboard is added to handle keyboard events.

```
class CExampleWindow: public CIvfWindow {
private:
    CIvfCamera*       m_camera;
    CIvfComposite*    m_scene;
    CIvfLight*        m_light;

    CIvfTexture*      m_logoTexture;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    int m_beginX;
    int m_beginY;

public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
    virtual void onKeyboard(int key, int x, int y);
};
```

In the onInit method the camera is placed at (0.0, 0.0, 6.0).

```
m_camera->setPosition(0.0, 0.0, 6.0);
```

A red material is created to be used when using texture blending.

```
CIvfMaterial* material = new CIvfMaterial();
material->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
material->setAmbientColor(0.3f, 0.3f, 0.3f, 1.0f);
```

The scene composite is created to manage the scene.

```
m_scene = new CIvfComposite();
```

The first step in using textures is to create the image used with the texture. An image can be created either by using an CIvfImage object directly, creating the image directly in code, or using the classes in the ivfimage library and loading the image from disk. In this example the image will be loaded from a .png image file using the CIvfPngImage class.

```
CIvfPngImage* logoImage = new CIvfPngImage();
logoImage->setFileName("images/ivf.png");
logoImage->read();
```

In the next step the CIvfTexture object is created. The previously created image object logoImage is then assigned to the texture.

```
m_logoTexture = new CIvfTexture();
m_logoTexture->setImage(logoImage);
```

When textures are enlarged or reduced different techniques can be used to magnify or reduce the pixels in the image. In the CIvfTexture object, the techniques are specified using the setFilters method. This method takes two parameters, the first specifying a minification filter and the second a magnification filter. The initial filter settings are set to GL_NEAREST for both magnification and minification. The initial generation of mipmaps are also enabled using the setGeneratMipmaps method. The filter and mipmap settings are explained later on in this example.

```
m_logoTexture->setFilters(GL_NEAREST, GL_NEAREST);
m_logoTexture->setGenerateMipmaps(true);
```

To display the texture a CIvfQuadPlane object is used. This is a simple geometry consisting of a single plane with texture coordinates. The material and textures created earlier are assigned to the object.

```
CIvfQuadPlane* logo = new CIvfQuadPlane();
logo->setSize(1.8, 1.8);;
logo->setMaterial(material);
logo->setTexture(m_logoTexture);

m_scene->addChild(logo);
```

Finally a directional light is created.

```
m_light = new CIvfLight();
m_light->setType(CIvfLight::LT_DIRECTIONAL);
m_light->setDirection(1.0, 1.0, 1.0);
m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
```

The onRender method is then modified to render the scene composite.

```
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}
```

In the onKeyboard keys are added to switch between different texture settings.

```
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    switch (key) {
    case '1':
        m_logoTexture->setFilters(GL_NEAREST,
            GL_NEAREST);
        break;
    case '2':
        m_logoTexture->setFilters(GL_LINEAR,
            GL_LINEAR);
        break;
    case '3':
        m_logoTexture->setFilters(
            GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
        break;
    case 'd':
        m_logoTexture->setMode(GL_DECAL);
        break;
    case 'b':
        m_logoTexture->setMode(GL_BLEND);
        break;
    case 'm':
        m_logoTexture->setMode(GL_MODULATE);
```

```
          break;
      default:

          break;
      }
      .
      .
```

Before these settings take effect the texture must be rebound using the bind method and the window redrawn.

```
      .
      .

      m_logoTexture->bind();

      redraw();
  }
```

The final application is shown in figure 4.11.

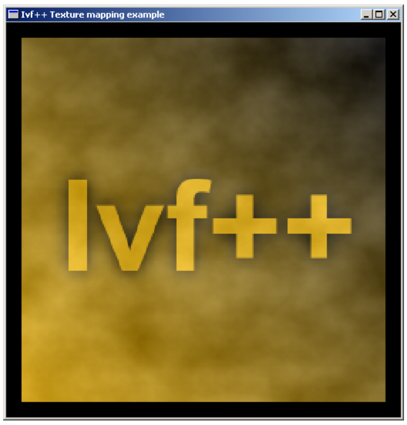


Figure 4.11: Texturing in Ivf++

As described earlier different filters can be using when a texture is magnified or reduced. Using the GL_NEAREST filter, the texture pixel nearest to the screen pixel, in magnification or minification. The GL_LINEAR filter will use a weighted average of the 2x2 array of image pixels nearest to the screen pixel. Closeups of a texture using these filters are shown in figure 4.12. The GL_LINEAR filter setting produces nice results when textures are magnified. By pressing '1' and '2' in these settings can be viewed.

When minification is used GL_LINEAR and GL_NEAREST can produce jitter especially when the camera or scene is moving. To reduce this, a technique call

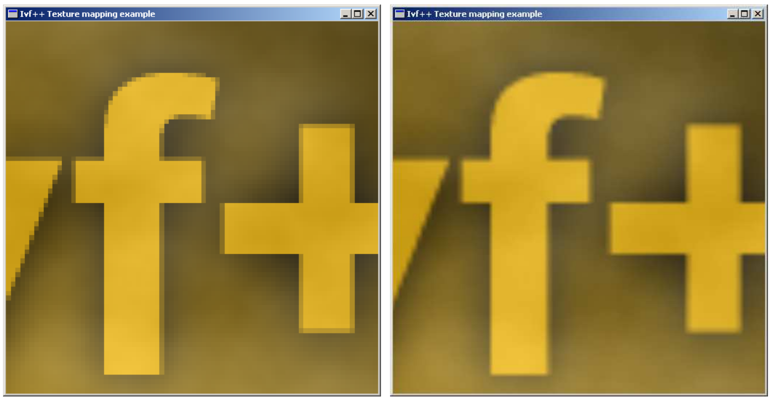


Figure 4.12: GL_NEAREST and GL_LINEAR filter settings

mipmapping can be used. In this technique a set of images are used from the original size downto 1x1 in steps of power 2. The largest will be used at close up and the 1x1 image when the texture has the size of a pixel. The other images will be chosen depending on the distance to the viewer. Mipmapping is enabled using the setGenerateMipmaps method in the CIvfTexture object. Selection of mipmap images is controlled using the minification filters.

Figure 4.13 illustrate the effects without mipmapping and with.

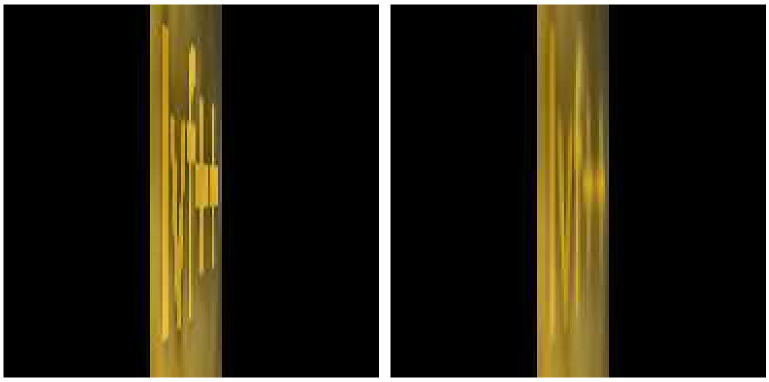


Figure 4.13: Texturing without mipmapping and with

Textures can be applied to the object in different ways. The default application is GL_DECAL. In this mode, the texture replaces all color information already defined with the object. GL_MODULATE will modulate the texture with the material colors or vertex colors. Finally, the GL_BLEND will use blending to combine the colors of the texture and the object colors. A more detailed description of these modes can be found in [7]. Setting the modes can be done using the setMode method of the *CIvfTexture* object. Figure 4.14 illustrate these settings.

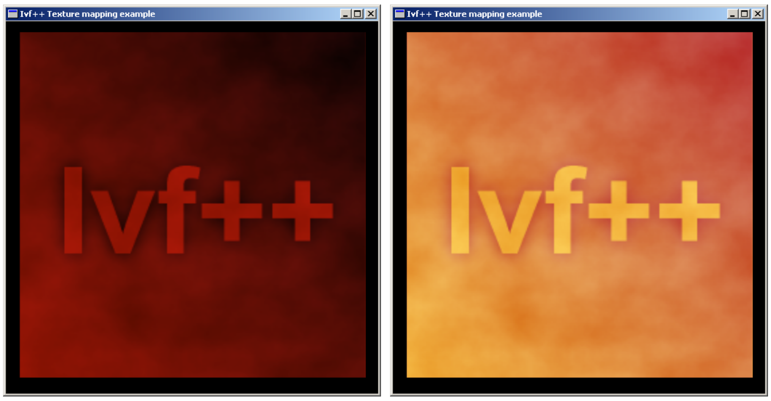


Figure 4.14: GL_MODULATE and GL_BLEND modes with a red material

In this example, use the keys 'd' for GL_DECAL, 'b' for GL_BLEND or 'm' for GL_MODULATE.

The source code of the program is shown in the next section.

4.5 Fonts

Ivf++ supports font rendering using the class ivffont library. This library us built on the FTGL and freetype libraries, supporting TrueType font rendering.

Chapter 5

Creating advanced geometry

5.1 Advanced geometry

To create more advanced geometry, Ivf++ supplies eight classes encapsulating the drawing primitives of OpenGL. All these classes are derived from CIvfGLPrimitive and have the same behaviour. They all contain a coordinate list, color list and a texture coordinate list. Geometry is described through the use of indices to the lists mentioned above. Figure 5.1 shows the relationships between the lists and indexes.

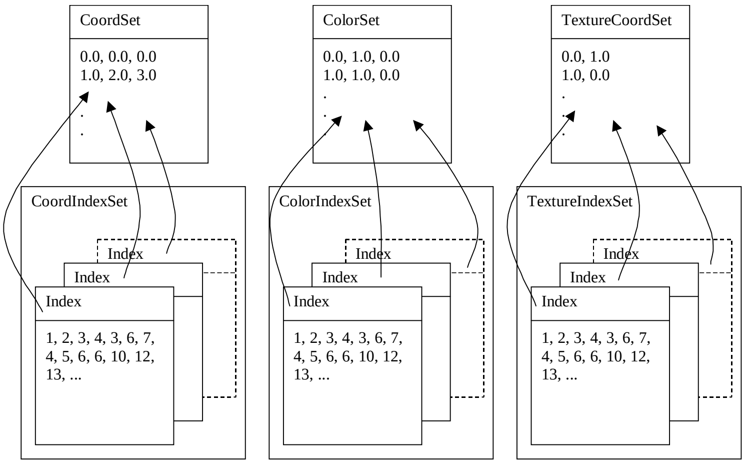


Figure 5.1: Index relationships

In this tutorial the use of the classes CIvfLineSet, CIvfLineStripSet, CIvfPointSet, CIvfTriSet, CIvfTriStripSet and CIvfQuadSet will be introduced. The includes for this example are shown below.

```
#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCylinder.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfTexture.h>

// CIvfGLPrimitive derived classes

#include <ivf/IvfPointSet.h>
#include <ivf/IvfLineSet.h>
#include <ivf/IvfLineStripSet.h>
#include <ivf/IvfTriSet.h>
#include <ivf/IvfTriStripSet.h>
#include <ivf/IvfQuadSet.h>

// From the image library

#include <ivfimage/IvfPngImage.h>
```

In the onInit, the camera is moved to (-6.6, 9.2, 10.5) and the target set to (1.1, 0.1, 0.6).

```
m_camera->setPosition(-6.6, 9.2, 10.5);
m_camera->setTarget(1.1, 0.1, 0.6);
```

Two materials are also created.

```
CIvfMaterial* greenMaterial = new CIvfMaterial();
greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f)
    ;
greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f
    );
greenMaterial->setAmbientColor(0.0f, 0.2f, 0.0f, 1.0f)
    ;

CIvfMaterial* redMaterial = new CIvfMaterial();
redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
redMaterial->setAmbientColor(0.2f, 0.0f, 0.0f, 1.0f);
```

A texture is also created, with an associated image.

```
// Create textures

CIvfPngImage* logoImage = new CIvfPngImage();
logoImage->setFileName("images/ivf.png");
logoImage->read();

CIvfTexture* logoTexture = new CIvfTexture();
logoTexture->setImage(logoImage);

// Create scene composite

m_scene = new CIvfComposite();
```

First a point set will be created. A point set corresponds to the OpenGL GL_POINT primitive. First the points are added.

```
CIvfPointSet* pointSet = new CIvfPointSet();

pointSet->addCoord(0.0, 0.0, 0.0);
pointSet->addCoord(0.0, 0.0, 1.0);
pointSet->addCoord(1.0, 0.5, 0.0);
pointSet->addCoord(1.0, 0.5, 1.0);
pointSet->addCoord(2.0, 0.5, 0.0);
pointSet->addCoord(2.0, 0.5, 1.0);
pointSet->addCoord(3.0, 0.0, 0.0);
pointSet->addCoord(3.0, 0.0, 1.0);
```

Even if a set of point doesn't have a topology, the drawn points must be referenced using an CIvfIndex class. The CIvfIndex class has several method for simplifying the creation of different types of indices. In this case all points are to be drawn, so a linear index is needed. The createLinear method of the CIvfIndex class can does this automatically.

```
CIvfIndex* coordIdx = new CIvfIndex();
coordIdx->createLinear(8);
```

The finished index is then added with the addCoordIndex method.

```
pointSet->addCoordIndex(coordIdx);
```

If the points have colors a set of colours are added in the same way as the coordinates given previously.

```
pointSet->addColor(0.0, 0.0, 1.0);
pointSet->addColor(0.0, 1.0, 0.0);
pointSet->addColor(0.0, 1.0, 1.0);
```

```
pointSet->addColor(1.0, 0.0, 0.0);
pointSet->addColor(1.0, 0.0, 1.0);
pointSet->addColor(1.0, 1.0, 0.0);
pointSet->addColor(1.0, 1.0, 1.0);
pointSet->addColor(0.0, 0.0, 1.0);
```

An index is created and added in the same way.

```
CIvfIndex* colorIdx = new CIvfIndex();
colorIdx->createLinear(8);

pointSet->addColorIndex(colorIdx);
```

To use the colors when drawing the point set the use color flag must be set using the setUseColor method.

```
pointSet->setUseColor(true);
```

If the use color flag is set to false, the material assigned to the object will be used. If no material is assigned the points will be drawn in a white color.

Finally the point set will be given a position, the point size will be set to 3 and it will be added to the scene composite.

```
pointSet->setPosition(-3.0, 0.0, 3.0);
pointSet->setPointSize(3);

m_scene->addChild(pointSet);
```

The resulting geometry is shown in Figure 5.2.

Next a CIvfLineSet derived object is created. This object handles a sets of lines defined as pairs of coordinates. The *CIvfLineStrip* object handles a set of continuous lines. Coordinates are added using the addCoord method. 8 coordinates are added.

```
CIvfLineSet* lineSet = new CIvfLineSet();

lineSet->addCoord(0.0, 0.0, 0.0);
lineSet->addCoord(0.0, 0.0, 1.0);
lineSet->addCoord(1.0, 0.5, 0.0);
lineSet->addCoord(1.0, 0.5, 1.0);
lineSet->addCoord(2.0, 0.5, 0.0);
lineSet->addCoord(2.0, 0.5, 1.0);
lineSet->addCoord(3.0, 0.0, 0.0);
lineSet->addCoord(3.0, 0.0, 1.0);
```

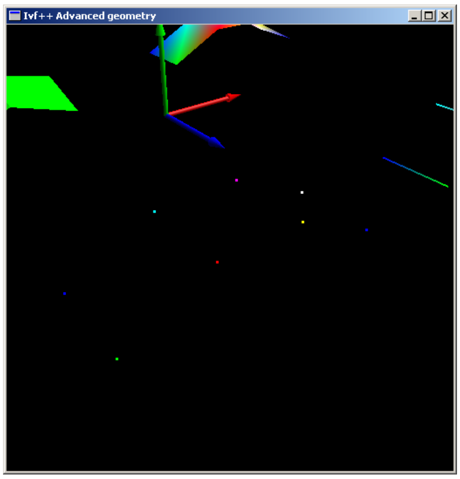


Figure 5.2: Resulting point set

The coordinates are be numbered as in figure 5.3.

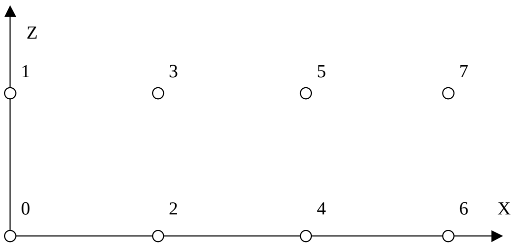


Figure 5.3: Coordinate numbering

To create the geometry an index of the lines must also be supplied. If the coordinates are added in ordered pairs, a linear index can be created. The following code is added to create and add an index to the line set.

```
coordIdx = new CIvfIndex();
coordIdx->createLinear(8);

lineSet->addCoordIndex(coordIdx);
```

Color will also be used supplied using the same methods as above, using the *addColor* instead of the *addCoord* method.

```
lineSet->addColor(0.0, 0.0, 1.0);
lineSet->addColor(0.0, 1.0, 0.0);
lineSet->addColor(0.0, 1.0, 1.0);
lineSet->addColor(1.0, 0.0, 0.0);
```

```
lineSet->addColor(1.0, 0.0, 1.0);
lineSet->addColor(1.0, 1.0, 0.0);
lineSet->addColor(1.0, 1.0, 1.0);
lineSet->addColor(0.0, 0.0, 1.0);

colorIdx = new CIvfIndex(); colorIdx->createLinear(8);

lineSet->addColorIndex(colorIdx);
```

To use only one color on all lines, the *createConstant* method of the *CIvfIndex* object can be used. The first parameter sets the value to assign all indices. The second sets the size of the index. Using the *createConstant* method the previous code changes to.

```
lineSet->addColor(1.0, 0.0, 0.0);

colorIdx = new CIvfIndex();
colorIdx->createConstant(0, 8);

lineSet->addColorIndex(colorIdx);
```

As in the point set the use color flag is set, the line set is positioned and added to the scene composite.

```
lineSet->setPosition(3.0, 0.0, 3.0);
lineSet->setUseColor(true);

scene->addChild(lineSet);
```

The resulting geometry that will be created when using the linear index is shown in Figure 5.4.

A CIvfLineStripSet object is created in the same way as in the previous examples:

```
CIvfLineStripSet* lineStripSet = new CIvfLineStripSet
    ();

lineStripSet->addCoord(0.0, 0.0, 0.0);
lineStripSet->addCoord(0.0, 0.0, 1.0);
lineStripSet->addCoord(1.0, 0.5, 0.0);
lineStripSet->addCoord(1.0, 0.5, 1.0);
lineStripSet->addCoord(2.0, 0.5, 0.0);
lineStripSet->addCoord(2.0, 0.5, 1.0);
lineStripSet->addCoord(3.0, 0.0, 0.0);
lineStripSet->addCoord(3.0, 0.0, 1.0);

coordIdx = new CIvfIndex();
coordIdx->createLinear(8);
```

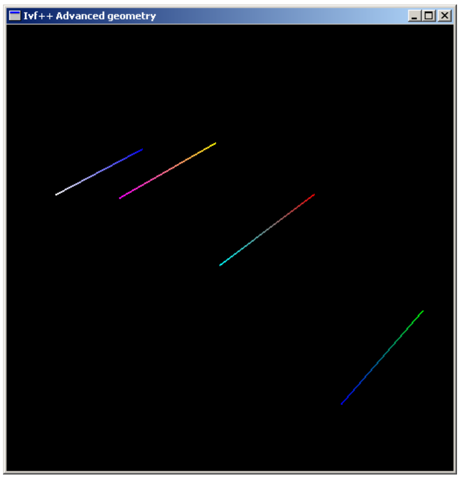


Figure 5.4: Lines using a linear index

```
lineStripSet->addCoordIndex(coordIdx);

lineStripSet->addColor(0.0, 0.0, 1.0);
lineStripSet->addColor(0.0, 1.0, 0.0);
lineStripSet->addColor(0.0, 1.0, 1.0);
lineStripSet->addColor(1.0, 0.0, 0.0);
lineStripSet->addColor(1.0, 0.0, 1.0);
lineStripSet->addColor(1.0, 1.0, 0.0);
lineStripSet->addColor(1.0, 1.0, 1.0);
lineStripSet->addColor(0.0, 0.0, 1.0);

colorIdx = new CIvfIndex();
colorIdx->createLinear(8);

lineStripSet->addColorIndex(colorIdx);

lineStripSet->setPosition(1.5, 3.0, 3.0);
lineStripSet->setUseColor(true);
lineStripSet->setLineWidth(2);

m_scene->addChild(lineStripSet);
```

The first part of creating a CIvfTriSet object is the same as creating CIvfLineSet object. 9 coordinates are added to the object.

```
CIvfTriSet* triSet = new CIvfTriSet();

triSet->addCoord(0.0,0.0,2.0);
triSet->addCoord(1.0,0.3,2.0);
```

```
triSet->addCoord(2.0,0.0,2.0);
triSet->addCoord(0.0,0.3,1.0);
triSet->addCoord(1.0,0.5,1.0);
triSet->addCoord(2.0,0.3,1.0);
triSet->addCoord(0.0,0.0,0.0);
triSet->addCoord(1.0,0.3,0.0);
triSet->addCoord(2.0,0.0,0.0);
```

A linear index can not be applied here because the orientation of the triangles are important and the above coordinate list is not ordered accordingly. The CIvfIndex object has a set of methods making it easier to add indices in a logical manner. The indices for each triangle in this case will be added three by three using the add method with three indices in each call. The add method can be supplied with 1-4 indices four each call. These add methods are just created as a convenience for the user of the library. Using the add method the code becomes.

```
coordIdx = new CIvfIndex();
coordIdx->add(0,1,4);
coordIdx->add(0,4,3);
coordIdx->add(1,2,5);
coordIdx->add(1,5,4);
coordIdx->add(3,4,7);
coordIdx->add(3,7,6);
coordIdx->add(4,5,8);
coordIdx->add(4,8,7);

triSet->addCoordIndex(coordIdx);
```

A set of texture coordinates will also be added.

```
triSet->addTextureCoord(0.0,0.0);
triSet->addTextureCoord(0.5,0.0);
triSet->addTextureCoord(1.0,0.0);
triSet->addTextureCoord(0.0,0.5);
triSet->addTextureCoord(0.5,0.5);
triSet->addTextureCoord(1.0,0.5);
triSet->addTextureCoord(0.0,1.0);
triSet->addTextureCoord(0.5,1.0);
triSet->addTextureCoord(1.0,1.0);
```

The texture coordinates are supplied in the same order as the coordinates, so the previous index ordering can be used. A new index will be created and the *coordIdx* index will be copied to the new index using the *assignFrom* method of the *CIvfIndex* object.

```
CIvfIndex* textureIdx = new CIvfIndex();
textureIdx->assignFrom(coordIdx);
```

The texture index is then added to the *CIvfTriSet* object using the *addTextureIndex* method.

```
triSet->addTextureIndex(textureIdx);
```

The CIvfTriSet object automatically generates surface normals making it possible to use lighting with the object. Default the object only generates surface normals. Using the setUseVertexNormals method vertex normals can also be generated. The CIvfTriSet object is positioned at (-3.0, 0.0, -3.0), given a material and texture, and finally the object is added to the scene.

```
triSet->setMaterial(greenMaterial);
triSet->setTexture(logoTexture);
//triSet->setUseVertexNormals(true);
triSet->setPosition(-3.0, 0.0, -3.0);

m_scene->addChild(triSet);
```

The resulting geometry is shown in figure 5.5.

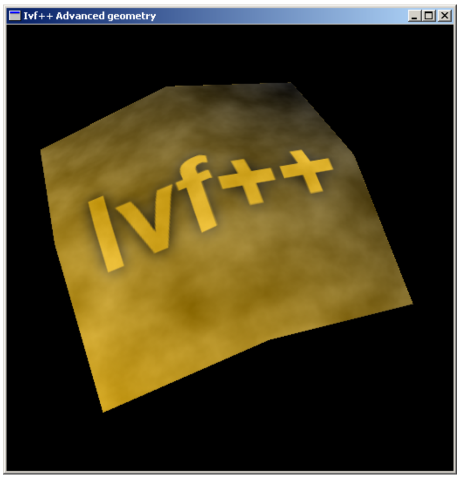


Figure 5.5: *CIvfTriSet* object

The CIvfQuadSet object creates four sided surfaces and can used in the same way as the CIvfTriSet object. To illustrate the use of a CIvfQuadSet object, a cube will be created with each corner in specified with a different color. The geometry is shown in Figure 5.6.

The coordinates are added with the following code.

```
CIvfQuadSet* quadSet = new CIvfQuadSet();
```

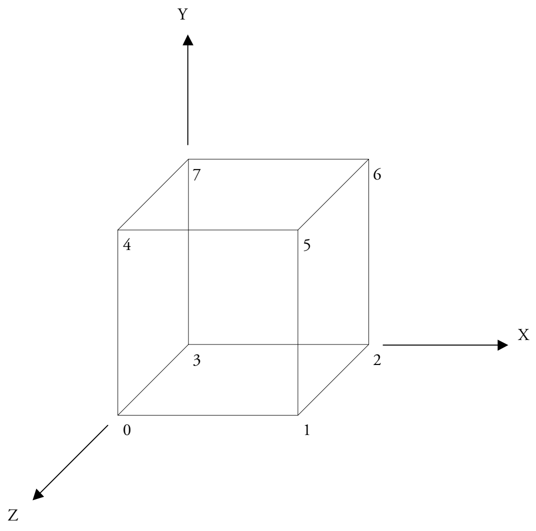


Figure 5.6: Cube geometry

```
quadSet->addCoord(0.0,0.0,1.0);
quadSet->addCoord(1.0,0.0,1.0);
quadSet->addCoord(1.0,0.0,0.0);
quadSet->addCoord(0.0,0.0,0.0);
quadSet->addCoord(0.0,1.0,1.0);
quadSet->addCoord(1.0,1.0,1.0);
quadSet->addCoord(1.0,1.0,0.0);
quadSet->addCoord(0.0,1.0,0.0);
```

The indices are added using the 4-paramter add method.

```
coordIdx = new CIvfIndex();
coordIdx->add(0,1,5,4);
coordIdx->add(1,2,6,5);
coordIdx->add(2,3,7,6);
coordIdx->add(3,0,4,7);
coordIdx->add(4,5,6,7);
coordIdx->add(0,3,2,1);

quadSet->addCoordIndex(coordIdx);
```

The same number of colors are added as there are coordinates. The index is created by copying it from the *coord* index using the *copyFrom* method of the *CIvfIndex* object.

```
quadSet->addColor(0.0, 0.0, 1.0);
quadSet->addColor(0.0, 1.0, 0.0);
```

```
quadSet->addColor(0.0, 1.0, 1.0);
quadSet->addColor(1.0, 0.0, 0.0);
quadSet->addColor(1.0, 0.0, 1.0);
quadSet->addColor(1.0, 1.0, 0.0);
quadSet->addColor(1.0, 1.0, 1.0);
quadSet->addColor(0.0, 0.0, 1.0);

colorIdx = new CIvfIndex();
colorIdx->assignFrom(coordIdx);

quadSet->addColorIndex(colorIdx);
```

The cube is placed at (-3.0, 3.0, -3.0) and the use color flag of the CIvfQuadSet is set to true to use the colors given above.

```
quadSet->setUseColor(true);
quadSet->setPosition(-3.0, 3.0, -3.0);

m_scene->addChild(quadSet);
```

The program window is shown in Figure 5.7.

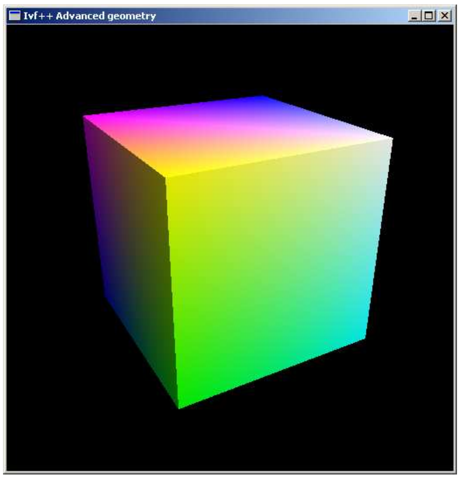


Figure 5.7: Colored cube using the CIvfQuadSet object

An axis object and a light are also created.

```
axis = new CIvfAxis();
axis->setSize(1.5);
scene->addChild(axis);

light = new CIvfLight();
light->setPosition(1.0, 1.0, 1.0, 0.0);
```

```
    light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
```

Before the program is finished the onRender method must be modified to render the scene composite.

The source code of the program is shown in the next section.

5.2 Extrusions

To create extruded shapes the GLE Tubing and Extrusion library [10] library can be used. The Ivf++ encapsulates some of this libraries functionality in the CIvfExtrusion object. In this tutorial, a simple extrusion will be created. The tutorial will be based on the code in section 3. An extrusion is created by supplying a set of points describing the spine of the extrusion and a set of points describing the section. The spine layout of this tutorial is given in Figure 5.8.

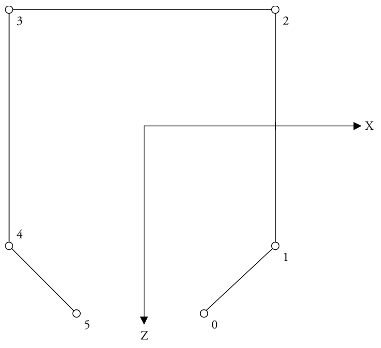


Figure 5.8: Spine geometry

The following includes are used.

```
    #include <ivfui/IvfApplication.h>
    #include <ivfui/IvfWindow.h>

    #include <ivf/IvfCamera.h>
    #include <ivf/IvfAxis.h>
    #include <ivf/IvfComposite.h>
    #include <ivf/IvfTransform.h>
    #include <ivf/IvfLight.h>
    #include <ivf/IvfMaterial.h>
    #include <ivf/IvfExtrusion.h>
```

In the onInit method a set of variables are defined for use in the creation of the section.

```
// Initialize variables

m_angleX = 0.0;
m_angleY = 0.0;
m_moveX = 0.0;
m_moveY = 0.0;
m_zoomX = 0.0;
m_zoomY = 0.0;

int i, nSides;
double r, angle, x, y;
```

The camera is created and positioned at (0.0, 5.0, 5.0).

```
camera = new CIvfCamera();
camera->setPosition(0.0, 5.0, 5.0);
```

A yellow material is created to be assigned to the extrusion object.

```
CIvfMaterial* yellowMaterial = new CIvfMaterial();
yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f
    );
yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0
    f);
yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f
    );

// Create scene composite

m_scene = new CIvfComposite();
```

The extrusion is created in the next step. Section points will be created first. Using to variables r and nSides the size and number of facets can be controlled. Before the section-points can be added, the section size must be set. This is done using the setSectionSize method of the CIvfExtrusion object.

```
// Create extrusion

CIvfExtrusion* extrusion = new CIvfExtrusion();

// Create section

r = 0.5;
nSides = 12;
```

```
extrusion->setSectionSize(nSides + 1);
```

Section coordinates are created by dividing a circle into nSides and adding to points to the extrusion object using the setSectionCoord method. The section normals are also calulated in the same loop and set with the setSectionNormal method.

```
for (i = 0; i<=nSides; i++)
{
    angle = 2.0*M_PI*( ((double)i) / ((double)nSides)
        );
    x = r * cos(angle);
    y = r * sin(angle);
    extrusion->setSectionCoord(i, x, y);
    extrusion->setSectionNormal(i, x/r, y/r);
}
```

Next, the points of the spine will be added. The first and last points of the spine will not be part of the visible extrusion; they are used to control the angles of the first and last sections. The number of spine points is set with the setSpineSize method and the points are the set using the setSpineCoord method.

```
extrusion->setSpineSize(6);
extrusion->setSpineCoord(0,   0.5,  0.0,  1.5);
extrusion->setSpineCoord(1,   1.0,  0.0,  1.0);
extrusion->setSpineCoord(2,   1.0,  0.0, -1.0);
extrusion->setSpineCoord(3,  -1.0,  0.0, -1.0);
extrusion->setSpineCoord(4,  -1.0,  0.0,  1.0);
extrusion->setSpineCoord(5,  -0.5,  0.0,  1.5);
```

Placement of the section along the axis of the extrusion is defined with the setUpVector method. This method sets the vector indicating the up-direction of the section.

```
extrusion->setUpVector(0.0, 1.0, 0.0);
```

The GLE library defines a set of constants controlling the extrusion appearance. These constants can be set by the setJoinStyle method. The following code tells the GLE library to generate edge normals, use angular joins and to cap the ends of the extrusion.

```
extrusion->setJoinStyle(TUBE_NORM_EDGE|TUBE_JN_ANGLE|
    TUBE_JN_CAP);
```

In the last step, a material is assigned to the extrusion and it is added to the scene.

```
extrusion->setMaterial(yellowMaterial);
m_scene->addChild(extrusion);
```

The resulting program window is shown in Figure 5.9.

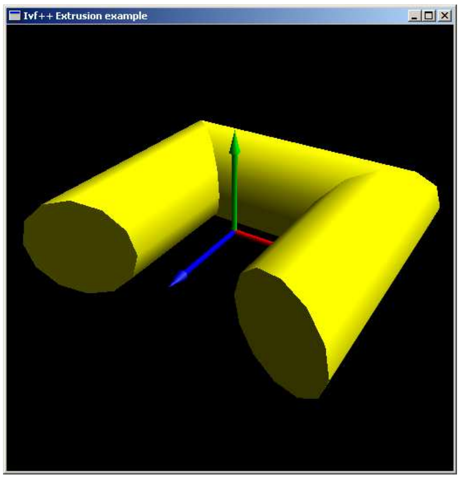


Figure 5.9: *CIvfExtrusion* object

The source code of the program is shown in the next section.

Chapter 6

Rendering techniques

6.1 Level of detail and switch objects

OpenGL is often used for real-time rendering of 3D graphics. This puts high demands on graphics hardware. There are limits of how many triangles that can be rendered every frame. To reduce the number of triangles that are rendered, different techniques can be used. Ivf++ implements a technique called level of detail or LOD. In this technique an object is represented a set of representations with decreasing triangle count. What representation will be rendered is determined by the distance to the viewer and transition function. The function used is shown in figure 6.1.

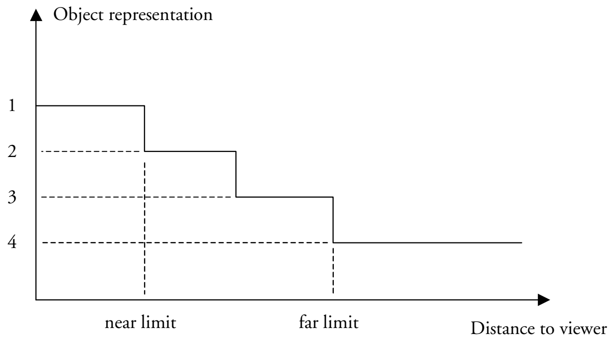


Figure 6.1: LOD transistion function

The CIvfLOD object is derived from the CIvfSwitch object, which is a CIvfComposite object showing only one of its children. This tutorial will create some spheres using the LOD technique and a switch object. The tutorial will be based on the code in chapter 3 and the onInit routine will be modified. The following includes are used.

```
#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfLOD.h>
#include <ivf/IvfCube.h>
#include <ivf/IvfCylinder.h>
```

In the class definition additional member variables for the sphere LOD objects and the switch objects are added. A private method updateLOD is added for updating the LOD limits for the spheres.

```
        .
        .
        // Ivf++ object declarations

        CIvfCamera*      m_camera;
        CIvfLight*       m_light;
        CIvfComposite*   m_scene;

        CIvfLOD*         m_lod1;
        CIvfLOD*         m_lod2;
        CIvfLOD*         m_lod3;
        CIvfLOD*         m_lod4;
        CIvfSwitch*      m_switch;

        double m_lodNear;
        double m_lodFar;

        void updateLOD();
public:
        CExampleWindow(int X, int Y, int W, int H)
                :CIvfWindow(X, Y, W, H) {};
        .
        .
```

In the onInit the member variables for controlling the LOD limits are initialised.

```
m_lodNear = 2.0; m_lodFar = 20.0;
```

The camera is then created and placed at (0.0, 5.0, 20.0).

```
m_camera = new CIvfCamera();
m_camera->setPosition(0.0, 5.0, 20.0);
```

A green material is created.

```
CIvfMaterial* material = new CIvfMaterial();
material->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
material->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);
```

The scene is created.

```
m_scene = new CIvfComposite();
```

Four different representations of a sphere are created with decreasing complexity. The complexity of a CIvfSphere object is controlled using the setStacks and setSlices methods.

```
CIvfSphere* sphereDetailed = new CIvfSphere();
sphereDetailed->setSlices(12);
sphereDetailed->setStacks(12);

CIvfSphere* sphereNormal = new CIvfSphere();
sphereNormal->setSlices(10);
sphereNormal->setStacks(8);

CIvfSphere* sphereSmall = new CIvfSphere();
sphereSmall->setSlices(8);
sphereSmall->setStacks(6);

CIvfSphere* sphereTiny = new CIvfSphere();
sphereTiny->setSlices(6);
sphereTiny->setStacks(4);
```

Each LOD object will use the above sphere representations. A CIvfLOD object is derived from a CIvfComposite class and objects are added using the addChild method. The first object added is the representation used when the viewer distance is less than the near limit in figure 6.1. A camera must also be assigned to the LOD object to be able to calculate the distance to the viewer. This is done using the setCamera method. LOD limits are specified using the setLimits method. In the following code we create the four LOD objects and assign camera and limits to all of them.

```
lod1 = new CIvfLOD(); lod1->addChild(sphereDetailed);
lod1->addChild(sphereNormal);
lod1->addChild(sphereSmall);
```

```
lod1->addChild(sphereTiny);
lod1->setPosition(-5.0, 0.0, -5.0);
lod1->setMaterial(material);
lod1->setCamera(camera);
lod1->setLimits(m_lodNear, m_lodFar);
scene->addChild(lod1);

lod2 = new CIvfLOD();
lod2->addChild(sphereDetailed);
lod2->addChild(sphereNormal);
lod2->addChild(sphereSmall);
lod2->addChild(sphereTiny);
lod2->setPosition(5.0, 0.0, -5.0);
lod2->setMaterial(material);
lod2->setCamera(camera);
lod2->setLimits(m_lodNear, m_lodFar);
scene->addChild(lod2);

lod3 = new CIvfLOD();
lod3->addChild(sphereDetailed);
lod3->addChild(sphereNormal);
lod3->addChild(sphereSmall);
lod3->addChild(sphereTiny);
lod3->setPosition(-5.0, 0.0, 5.0);
lod3->setMaterial(material);
lod3->setCamera(camera);
lod3->setLimits(m_lodNear, m_lodFar);
scene->addChild(lod3);

lod4 = new CIvfLOD();
lod4->addChild(sphereDetailed);
lod4->addChild(sphereNormal);
lod4->addChild(sphereSmall);
lod4->addChild(sphereTiny);
lod4->setPosition(5.0, 0.0, 5.0);
lod4->setCamera(camera);
lod4->setMaterial(material);
lod4->setLimits(m_lodNear, m_lodFar);
scene->addChild(lod4);
```

To illustrate the use of switch objects a cylinder and cube is created and added to a switch object.

```
CIvfCube* cube = new CIvfCube();
cube->setMaterial(material);
CIvfCylinder* cylinder = new CIvfCylinder();
cylinder->setMaterial(material);

m_switch = new CIvfSwitch();
m_switch->addChild(cube);
```

```
    m_switch->addChild(cylinder);

    m_scene->addChild(m_switch);
```

The finished program is shown in figures 6.3.

A special routine updateLOD is added to control the limits of all LOD objects.

```
void CExampleWindow::updateLOD()
{
    m_lod1->setLimits(m_lodNear, m_lodFar);
    m_lod2->setLimits(m_lodNear, m_lodFar);
    m_lod3->setLimits(m_lodNear, m_lodFar);
    m_lod4->setLimits(m_lodNear, m_lodFar);
}
```

Additional keys are defined in the keyboard callback to modify the LOD limits. Switching between the children in the CIvfSwitch object can be done using the cycleForward, cycleBackwards and setCurrentChild. Pressing the 'd' key will cycle to the next child in the *CIvfSwitch* object.

```
void CExampleWindow::onKeyboard(int key)
{
    switch (key) {
    case 'a':
        m_lodNear += 0.5;
        updateLOD();
        redraw();
        break;
    case 'z':
        m_lodNear -= 0.5;
        updateLOD();
        redraw();
        break;
    case 's':
        m_lodFar += 0.5;
        updateLOD();
        redraw();
        break;
    case 'x':
        m_lodFar -= 0.5;
        updateLOD();
        redraw();
        break;
    case 'd':
        m_switch->cycleForward();
        redraw();
        break;
    default:
```

```
                break;
            }
        }
```

The resulting program window at this point is shown in figure 6.2.

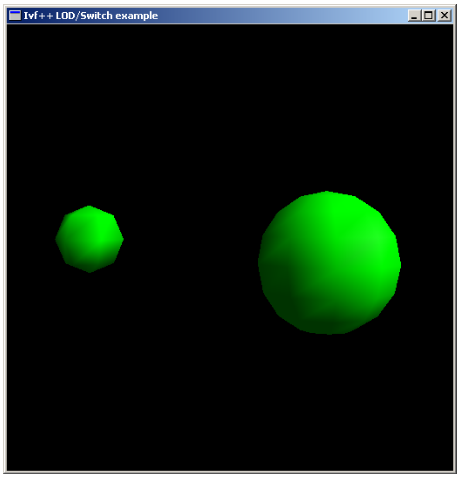


Figure 6.2: Objects with different LOD-shapes

When zooming in and out of the scene the LOD objects will automatically change their representations according to the distance to the viewer. The 'a' and 'z' keys moves the near limit forwards and backwards. The 's' and 'x' keys move the far limit in the same way.

The source code of the program is shown in the next section.

6.2 Scene graph culling

Often when viewing 3D scenes much of the geometry is not visible in the view window. OpenGL [1] automatically culls away this geometry in the rendering pipeline, but valuable time is lost sending not visible geometry to the 3D hardware. A way to solve this is to cull away geometry before it is sent to the graphics hardware. This is often done using a technique called scene graph culling. This technique uses a concept of bounding spheres. A bounding sphere is a sphere that contains the object representation. Instead of culling objects on a triangle level as in OpenGL spheres representing the objects are tested against the view volume. Spheres are easy to handle geometrically and an algorithm for determining if a sphere is inside the view volume can be implemented effectively. All CIvfShape derived classes can handle their bounding spheres. The culling algorithm is implemented in the CIvfCulling class.

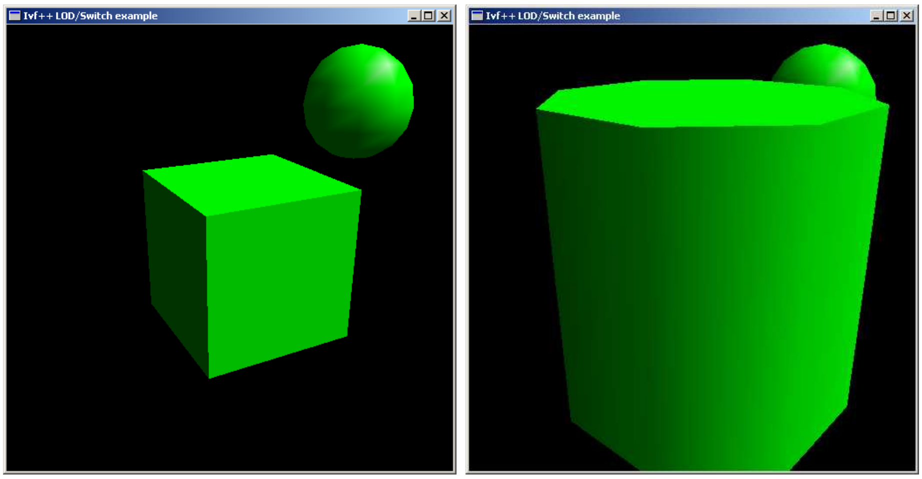


Figure 6.3: Switch states

The tutorial will be based on the code in chapter 3 and the onInit routine will be modified. The following includes are used.

```
#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCulling.h>
```

To illustrate the scene culling algorithm two cameras will be used. One camera will be the standard camera moving in the scene, the second camera will be an external camera observing the visible objects from above. The cameras are declared as member variables in the program. The *CIvfCulling* algorithm object is also added as a member variable.

```
// Ivf++ object declarations

CIvfCamera*         m_camera;
CIvfCamera*         m_externalCamera;
CIvfComposite*      m_scene;
CIvfCulling*        m_culling;
CIvfLight*          m_light;
CIvfCamera*         m_currentCamera;
```

```
        bool m_useCulling;

    public:
        CExampleWindow(int X, int Y, int W, int H)
            :CIvfWindow(X, Y, W, H) {};

        .
        .
```

In the next steps the onInit method will be modified. First the member variable m_useCulling is set to true to control culling in the example.

```
    m_useCulling = true;
```

Next the two cameras are created. The standard camera m_camera is placed in the middle of the scene, and the m_externalCamera is placed away from the scene.

```
    m_camera = new CIvfCamera();
    m_camera->addReference();
    m_camera->setPosition(0.0, 0.0, 10.0);

    m_externalCamera = new CIvfCamera();
    m_externalCamera->addReference();
    m_externalCamera->setPosition(25.0, 50.0, 50.0);
```

A special member variable m_currentCamera is used handle the switching of the cameras.

```
    m_currentCamera = m_camera;
```

A a simple green material is created. This material will be used to color the objects later on.

```
    CIvfMaterial* material = new CIvfMaterial();
    material->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    material->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);
```

The scene composite is created.

```
    m_scene = new CIvfComposite();
```

The geometry created in this tutorial will consist of a 3D array of spheres. To save memory the sphere geometry will be reused and a *CIvfTransform* objects will be used to place the spheres. The geometry setup code is shown below.

```
int i, j, k;
int nNodes = 8;
double distance = 20.0/(nNodes-1);

// Geometry is reduced by reusing the same sphere.

CIvfSphere* sphere = new CIvfSphere();
sphere->setSlices(12);
sphere->setStacks(12);
sphere->setMaterial(material);
sphere->setRadius(0.5);

// Create a grid of spheres.

CIvfTransform* arrayXlt = new CIvfTransform();
CIvfTransform* xlt;

for (i=0; i<nNodes; i++)
    for (j=0; j<nNodes; j++)
        for (k=0; k<nNodes; k++)
        {
            xlt = new CIvfTransform();
            xlt->setPosition(
                -10 + distance*i,
                -10 + distance*j,
                -10 + distance*k);
            xlt->addChild(sphere);
            arrayXlt->addChild(xlt);
        }

arrayXlt->setPosition(0.0, 0.0, -15.0);
arrayXlt->setRotationQuat(0.0, 0.0, 1.0, 45.0);

m_scene->addChild(arrayXlt);
```

In the next step the CIvfCulling algorithm object is created. To use the CIvfCulling object it must be assigned a CIvfComposite object containing the scene graph and a view used to cull the scene with.

```
m_culling = new CIvfCulling();
m_culling->setComposite(m_scene);
m_culling->setCullView(m_camera);
```

Finally a light is created.

```
m_light = new CIvfLight();
m_light->setType(CIvfLight::LT_DIRECTIONAL);
```

```
    m_light->setDirection(1.0, 1.0, 1.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
```

The onRender method is modified to support culling and switching of cameras. Culling is done using the cull of the CIvfCulling class.

```
void CExampleWindow::onRender()
{
    if (m_useCulling)
        m_culling->cull();

    m_light->render();
    m_currentCamera->render();
    m_scene->render();

    cout << "Culled objects = " << m_culling->
        getCullCount() << endl;;
}
```

To test the effect of scene culling the onKeyboard method is modified to support two keyboard commands [c] and [x]. [c] will turn on and off scene graph culling. [x] will switch between the two cameras.

```
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    switch (key) {
    case 'c':
        if (m_useCulling)
        {
            cout << "culling off" << endl;
            m_useCulling = false;
        }
        else
        {
            cout << "culling on" << endl;
            m_useCulling = true;
        }

        redraw();
        break;
    case 'x':
        if (m_currentCamera==m_camera)
        {
            m_currentCamera = m_externalCamera;
        }
        else
        {
            m_currentCamera = m_camera;
        }
```

```
            redraw();
            break;
        default:
            break;
        }
    }
```

The last thing to be modified is the onDestroy routine. The cameras and the scene has to be destroyed.

```
    void CExampleWindow::onDestroy()
    {
        delete m_light;
        delete m_culling;
        delete m_scene;
        delete m_camera;
        delete m_externalCamera;
    }
```

The figure 7.2 shows the culled view from the standard camera.

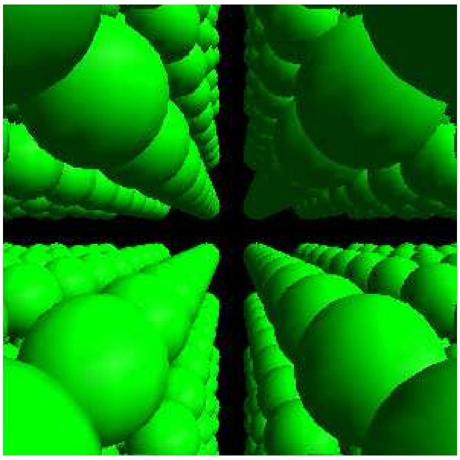


Figure 6.4: Culled scene from standard camera

The external camera view unculled and culled is shown in figure 6.5.

The source code of the program is shown in the next section.

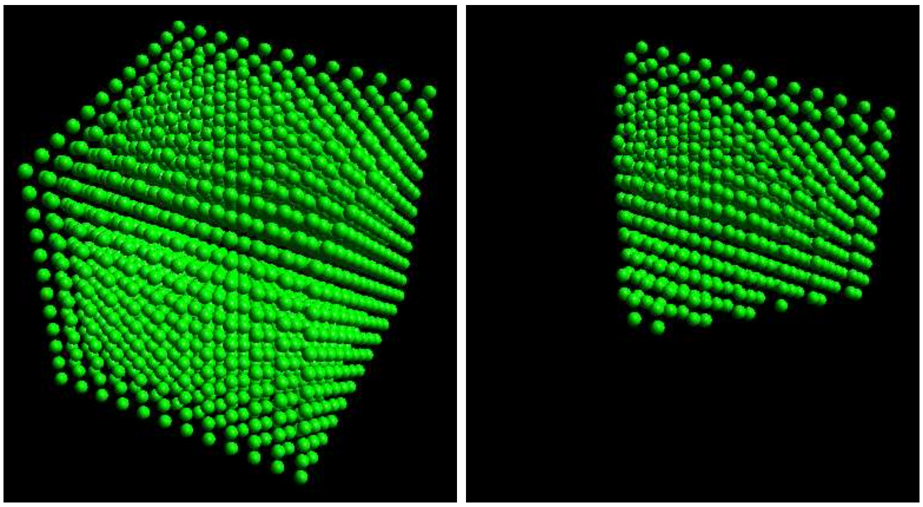


Figure 6.5: Unculled and culled scene

Chapter 7

Advanced techniques

7.1 Object selection

Selecting objects is often of importance. To support selection Ivf++ provides the CIvfBufferSelection class. This class encapsulates the OpenGL [1] selection process. Selection is done just by providing the mouse coordinates. The selection composite then returns a pointer to the selected object. To be able to handle selection the selection composite must be assigned a view. To illustrate the selection process the tutorial in from section 3.1 modified to support selection. The following includes are used:

```
#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfCube.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCylinder.h>
#include <ivf/IvfCone.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfBufferSelection.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
```

The CIvfBufferSelection algorithm object is added to the class definition.

```
    .
    .
    CIvfCamera*             m_camera;
    CIvfComposite*          m_scene;
    CIvfLight*              m_light;
```

```
        CIvfBufferSelection*      m_selection;
    public:
        CExampleWindow(int X, int Y, int W, int H)
            :CIvfWindow(X, Y, W, H) {};
        .
        .
```

The camera is placed at (0.0, 4.0, 9.0).

```
    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 4.0, 9.0);
```

A simple scene of objects is created.

```
    // Create a materials

    CIvfMaterial* redMaterial = new CIvfMaterial();
    redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    redMaterial->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

    CIvfMaterial* greenMaterial = new CIvfMaterial();
    greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f)
        ;
    greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f
        );
    greenMaterial->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f)
        ;

    CIvfMaterial* blueMaterial = new CIvfMaterial();
    blueMaterial->setDiffuseColor(0.0f, 0.0f, 1.0f, 1.0f);
    blueMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f)
        ;
    blueMaterial->setAmbientColor(0.0f, 0.0f, 0.5f, 1.0f);

    CIvfMaterial* yellowMaterial = new CIvfMaterial();
    yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f
        );
    yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0
        f);
    yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f
        );

    // Create scene composite

    m_scene = new CIvfComposite();

    // Create objects
```

```
CIvfCube* cube = new CIvfCube();
cube->setMaterial(redMaterial);
cube->setPosition(2.0, 0.0, 2.0);
m_scene->addChild(cube);

CIvfSphere* sphere = new CIvfSphere();
sphere->setMaterial(greenMaterial);
sphere->setPosition(-2.0, 0.0, 2.0);
m_scene->addChild(sphere);

CIvfCylinder* cylinder = new CIvfCylinder();
cylinder->setMaterial(blueMaterial);
cylinder->setPosition(-2.0, 0.0, -2.0);
m_scene->addChild(cylinder);

CIvfCone* cone = new CIvfCone();
cone->setMaterial(yellowMaterial);
cone->setPosition(2.0, 0.0, -2.0);
cone->setRotationQuat(0.0, 0.0, 1.0, 45.0);
m_scene->addChild(cone);

CIvfAxis* axis = new CIvfAxis();
m_scene->addChild(axis);
```

The CIvfBufferSelection algorithm is created in the next step. The algorithm needs a CIvfComposite object of the scene graph used in the selection process and the current view.

```
m_selection = new CIvfBufferSelection();
m_selection->setView(m_camera);
m_selection->setComposite(m_scene);
```

A light is also created.

```
m_light = new CIvfLight();
m_light->setType(CIvfLight::LT_DIRECTIONAL);
m_light->setDirection(1.0, 1.0, 1.0);
m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
```

The onMouseDown method is modified to do the selection. The selection is initiated by the pick method, which takes as input the x and y coordinates of the current mouse position.

```
void CExampleWindow::onMouseDown(int x, int y)
{
    if (isLeftButtonDown())
    {
```

```
                m_selection->pick(x, y);
                .
                .
```

The selected shape is return using the getSelectedShape method. In this example, if getSelectedShape returns a shape, it will be highlighted using the setHightlight method. Otherwise the highlight is removed from all objects in the scene.

```
                .
                .

                if (m_selection->getSelectedShape()!=NULL)
                {
                    m_selection->getSelectedShape()->
                        setHighlight(IVF_HIGHLIGHT_ON);
                    redraw();
                }
                else
                {
                    m_scene->setHighlightChildren(
                        IVF_HIGHLIGHT_OFF);
                    redraw();
                }
            }
        }
```

Objects can now be highlighted by clicking in the window. The program window is shown in Figure 7.1 with a highlighted object.

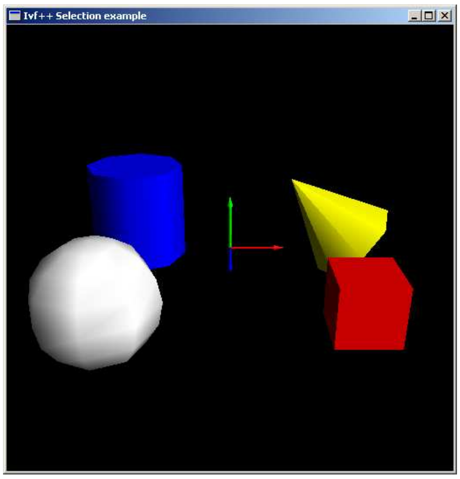


Figure 7.1: Scene with sphere selected

The source code of the program is shown in the next section.

7.2 Loading 3d models

It is often useful to be able to load 3d models created in other applications. Ivf++ supports the following 3d file formats:

- AC3D (.ac) most features supported.
- Drawing Exchange Format (.dxf) 3DFACE loading
- Poly (.ply) loading of surfaces.

The AC3D format was chosen because the file format has a simple structure and the AC3D [?] program can import many other formats. 3DStudio [?] will be supported in coming version using the lib3ds [?] library.

In this example a AC3D file will be loaded and rendered. To use the Ivf++ file loaders the ivffile library is used. The includes are shown below.

```
#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfLight.h>

#include <ivffile/IvfAc3DReader.h>
```

To use the AC3D reader, an CIvfAc3DReader object is created in the onInit method.

```
CIvfAc3DReader* acReader = new CIvfAc3DReader();
```

In the next step the filename is assigned to the file reader object and the file is read using the read method.

```
// Set parameters

acReader->setFileName("models/groups.ac");
acReader->setScaling(1.0);

// Read file

acReader->read();
```

To retrieve the read geometry the getShape method is used. This returns a pointer to a CIvfShape derived object. It is important to understand that the read shape is owned by the CIvfAc3DReader object. If the reader is deleted and the read shape is unreferenced, it will be deleted by the reader. In this example the read shape is added to the scene before the reader is deleted.

```
// Retrieve poly set

CIvfShape* shape = acReader->getShape();

m_scene->addChild(shape);

delete acReader;
```

Figure **??** shows a AC3D model loaded in Ivf++.

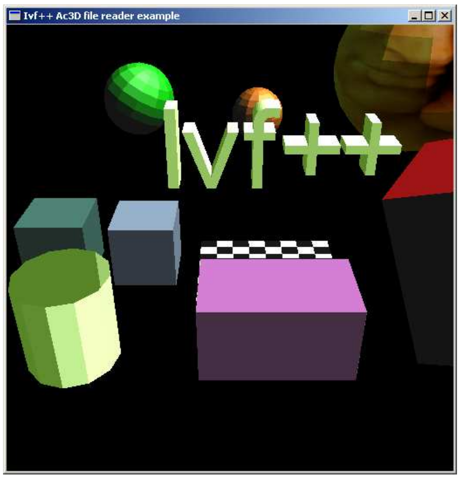


Figure 7.2: AC3D model viewed in Ivf++

he source code of the program is shown in the next section.

Chapter 8

Using Ivf++ with the FLTK library

The FLTK (Fast Light Toolkit) [4] is a user interface library that can be used both on UNIX and Windows based systems. To be able to use the Ivf++ library together with the FLTK library a special widget class has been created CIvfFltkBase. The class should not be used directly. Users of the class should derive a new class and implement the abstract virtual methods given in the CIvfFltkWidget class. The widget class has lot of methods and it is not possible at this time document the entire class, instead a simple example is given to show the working principle of the widget class. The finished application is shown in figure 8.1

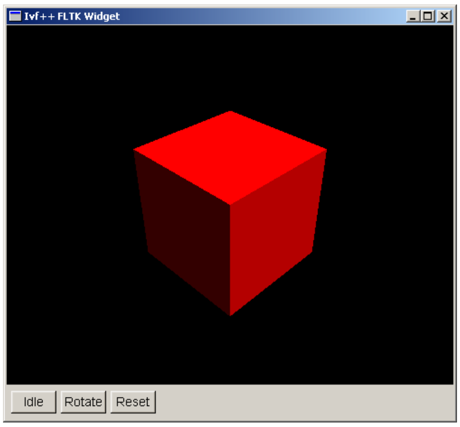


Figure 8.1: Finished FLTK application with Ivf++ widget

The first step to create this application is to create the main window using the FLUID user interface designer. FLUID uses special .fl-files to store user interface settings. From these files C++ code is generated. FLUID is started from the command line by entering the command **fluid**. The FLUID window is shown in Figure 8.2

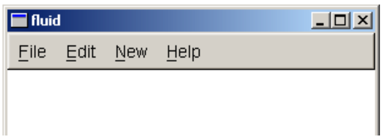
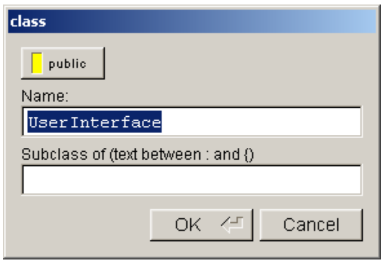


Figure 8.2: FLUID user interface designer

To create a new project in FLUID, the menu File/New is chosen. A file dialog appears and a filename is requested. MainFrame.fl is chosen as filename for the main window. FLUID is a class centered user interface designer. That means that a class is visually created in the FLUID main window. To create the single window for the application a class has to be created. This is done by selecting the menu New/Code/Class. In the dialog the name of the class is entered and the class it is derived from. See figure 8.3. In this case our class is not derived from any subclass. The class is named CMainFrame.


class

public

Name:

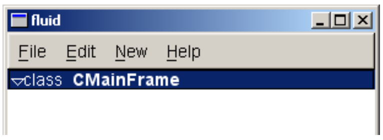
UserInterface

Subclass of (text between : and {)

OK Cancel

Figure 8.3: New class dialog

The FLUID window now shows the name of the newly created class.

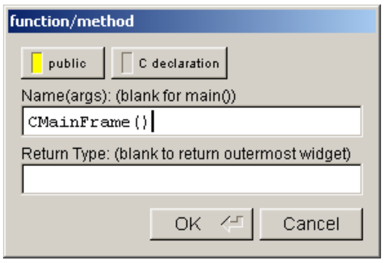


class CMainFrame

Figure 8.4: FLUID with the new class CMainFrame

In the next step the class constructor is created. This is done by selecting New/code/function/method in the menu. The dialog is shown in figure 8.5

In the dialog the name of the method is entered. In this case the name of the class and no parameters. The FLUID window now shows the class and its constructor in the hierarchical tree view. See figure 8.6.


function/method

public

C declaration

Name(args): (blank for main())

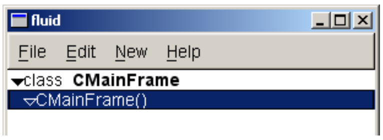
CMainFrame()

Return Type: (blank to return outermost widget)

OK

Cancel

Figure 8.5: Adding a class constructor



class CMainFrame
CMainFrame()

Figure 8.6: Class constructor in the FLUID designer

In the class constructor all controls will be initialized. This is done automatically by FLUID when controls are created.

Next the controls are created. First the the window is created. The window will hold all additional controls that will be added. The window control is added by selecting New/Group/Window in the menu. This shows the dialog in figure 8.7.

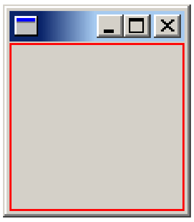
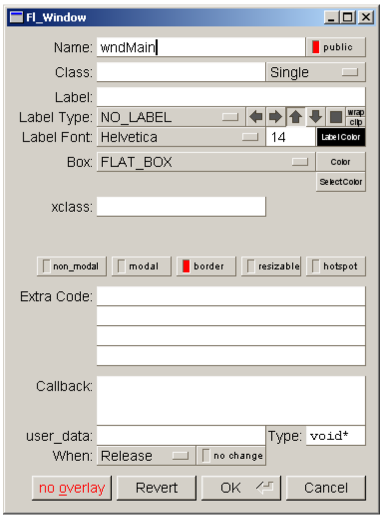


Figure 8.7: The default window control

The window control can be resized by dragging its borders. The properties of the window is changed by double-clicking on the window or the item in the FLUID window. This brings up the properties window as shown in figure 8.8

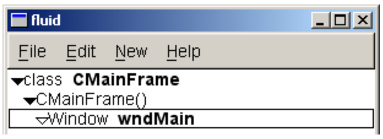
In the properties window the name of the window is set towndMain. This is the name the control will get when the C++ code is generated and is also used when referencing it from code written in the FLUID designer. The window label is set to ”Ivf++ FLTK application” and the resizable button is checked, to make the window resizable. The updated tree-view of the FLUID main window is shown in figure 8.9



Fl_Window

Name: wndMain public
Class: Single
Label:
Label Type: NO_LABEL
Label Font: Helvetica 14
Box: FLAT_BOX
xclass:
non_modal modal border resizable hotspot
Extra Code:
Callback:
user_data: Type: void*
When: Release no change

Figure 8.8: Window properties



class CMainFrame
CMainFrame()
Window wndMain

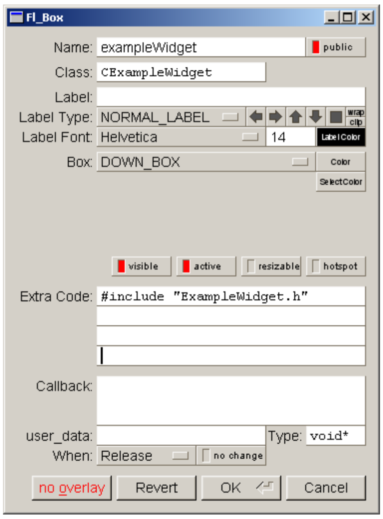
Figure 8.9: FLUID window with window control

The CIvfFltkBase derived widget will now be added to the window control. FLUID does not know anything about the CIvfFltkBase, therefore a special trick is used to add it to the window. A box is created by selecting New/Other/Box in the menu. The properties of the box is shown by double-clicking on it in the window. A property window for the box control is shown. The modified properties for the box is shown in figure 8.10.

Later on the CExampleWidget class will be created. For now it important that the name and class properties are correct and that the include directive is entered in the "Extra Code" box. The box control is also given a "sunken" appearance by changing the "Box" property to DOWN_BOX. The box control is resized to fill all but the bottom 10 percent of the window. See figure 8.11.

In the bottom part of the window the buttons are placed. This is done by selecting New/Buttons/Button in the menu. The first button added will be the "Idle" button. In the properties window the button is given the name "btnIdle" and the label set to "Idle". In the callback part of the properties window, the code that is executed when the control is pressed is placed. See figure 8.12.

When the "Idle" button is pressed, the idle processing in the CExampleWidget



```
Name: exampleWidget          public
Class: CExampleWidget
Label:
Label Type: NORMAL_LABEL
Label Font: Helvetica   14
Box: DOWN_BOX
visible  active  resizable  hotspot
Extra Code: #include "ExampleWidget.h"
Callback:
user_data:              Type: void*
When: Release    no change
```

Figure 8.10: Properties for the "box" control

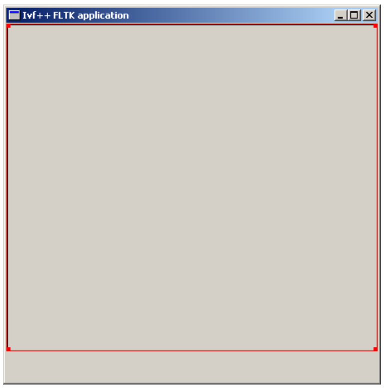
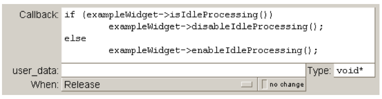


Figure 8.11: Resized box-control



```
Callback: if (exampleWidget->isIdleProcessing())
                  exampleWidget->disableIdleProcessing();
          else
                  exampleWidget->enableIdleProcessing();
user_data:                         Type: void*
When: Release          no change
```

Figure 8.12: Callback code for the "Idle" button

is either activated or deactivated.

A "Rotate" button will also be created. In the FLUID designer the button is given the name "btnRotate". In the callback the timer callback 0 is enabled and disabled.

```
if (exampleWidget->isTimeoutEnabled(0))
    exampleWidget->disableTimeout(0);
else
    exampleWidget->enableTimeout(0.1, 0);
```

The final button ”Reset” will be used to reset the camera to the default position, as in the onKeyboard method in example in chapter 3. The callback calls a special method in the CExampleWidget class, resetCamera.

Note also that it is imported the window-control is selected in the FLUID window to place the buttons on the window control instead of the box-control. When all the buttons are created the window control should look like figure 8.13.

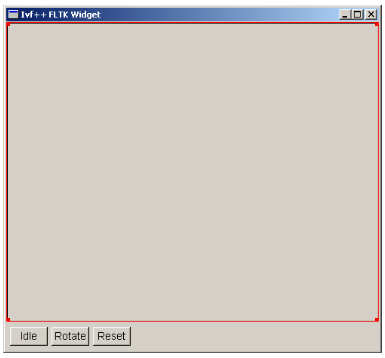
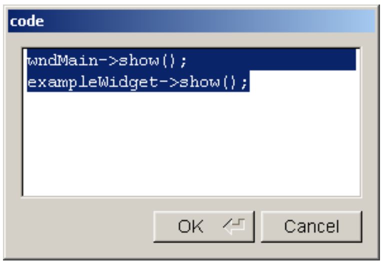


Figure 8.13: Finished application window

Before the code is generated an additional method for showing the window on the screen has to be added. The show method is added by selecting New/Code/-function/method in the menu. ”show()” is entered in the name textbox of the displayed dialog. To add code to the method New/Code/code is selected in the menu. The code entered is shown in figure 8.14.



```
wndMain->show();
exampleWidget->show();
```

Figure 8.14: Code editor

The complete user interface is now complete. The .fl-file is saved by selecting File/Save in the menu. The final step is to create the C++ code that is going to be used in the program. First the code generation properties is set by selecting Edit/Preferences. In the "Output File Names:" group the code file extension is changed to .cpp. The code is then generated by selecting File/Write code in the menu. The FLUID designer will generate two files a MainFrame.h and MainFrame.cpp. These files are then included in the makefile or project-file. In the next step the customized widget class CExampleWidget will be implemented. The event methods onInit, onDestroy, onResize, onKeyboard and onMouseDown from the CIvfFltkBase class are overridden. The onInit event method is used to initialize variables and Ivf++ objects. Initializing Ivf++ objects in the class constructor can be problematic if no OpenGL context is available. The onInit method is called the first time a window is shown and a current OpenGL contexts exists. The onDestroy is called before the widget is destroyed. The CExampleWidget class header is shown in the code below.

```
#ifndef _CExampleWidget_h_
#define _CExampleWidget_h_

#include <ivffltkwidget/IvfFltkBase.h>

#include <ivf/IvfComposite.h>
#include <ivf/IvfCamera.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfCube.h>

class CExampleWidget : public CIvfFltkBase {
private:

    // Camera movement state variables

    int m_beginX;
    int m_beginY;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    CIvfCamera*     m_camera;
    CIvfCube*       m_cube;
    CIvfLight*      m_light;
    CIvfComposite*  m_scene;
public:
    CExampleWidget(int X, int Y, int W, int H, const
        char *L=0);

    // Method used to reset the camera

    void resetCamera();
```

```
        // Basic ivfui event methods

        virtual void onInit(int width, int height);
        virtual void onResize(int width, int height);
        virtual void onRender();
        virtual void onDestroy();

        // Mouse event methods

        virtual void onMouseDown(int x, int y);
        virtual void onMouseMove(int x, int y);
        virtual void onMouseUp(int x, int y);

        // Keyboard event methods

        virtual void onKeyboard(int key, int x, int y);

        // Idle processing

        virtual void onIdle();

        // Timer processing

        virtual bool onTimeout0();
};
#endif
```

Additional member variables are used to set the node and line geometry. Materials for the nodes and lines are also defined. The constructor of the CExampleWidget just calls constructor of the Fl_Gl_Window class. In the onInit method all Ivf++ objects and variables are initialized. The implementation of the CExampleWidget class is equivalent of the example given in the chapter 3, except for the resetCamera method. This method resets the camera to the default position and is used to reset the camera from the "Reset"-button.

```
void CExampleWidget::resetCamera()
{
        m_camera->setPosition(2.0, 2.0, 2.0);
        m_camera->setTarget(0.0, 0.0, 0.0);
        this->redraw();
}
```

The rudimentary widget control is now complete. The final step will be to create a main routine. In the main routine the CMainFrame class will be instantiated and displayed using the show method. The display mode of the FLTK library will also be set. The main routine is shown below.

```
#include <FL/Fl.H>
#include "MainFrame.h"

int main(int argc, char **argv)
{
    Fl::visual(FL_DOUBLE|FL_RGB);
    Fl::get_system_colors();

    CMainFrame *frame = new CMainFrame();
    frame->show();

    return Fl::run();
}
```

Widget controls can be created for most of the GUI libraries available. The ivfwidget library contains abstract base classes, which contain basic widget functionality. The Ivf++ library has been used in the Gtk+ [6] OpenGL widget and as an MFC View on Windows based systems.

Chapter 9

Extending the Ivf++ library

There a number of different ways to extend the Ivf++ library, many of them will not be mentioned here. Three main approaches are illustrated in the following sections.

9.1 Derived composite classes

The easiest way to extend the Ivf++ library is by building composite classes. These classes will be derived from *CIvfComposite* and create an aggregate of objects. In the constructor of the new class all objects used in the new class are instantiated and added to the composite. The following code shows an example of how a composite class is defined. Class header:

```
#ifndef _CIvfCubeSphere_h_
#define _CIvfCubeSphere_h_

#include <ivf/IvfComposite.h>

 class CIvfCubeSphere : public CIvfComposite {
 public:
     CIvfComposite();
 };

#endif
```

Class implementation:

```
#include "IvfCubeSphere.h"

#include <ivf/IvfSphere.h>
#include <ivf/IvfCube.h>
```

```
CIvfCubeSphere::CIvfCubeSphere()
    :CIvfComposite()
{
    CIvfSpherePtr sphere = new CIvfSphere();
    CIvfCubePtr cube = new CIvfCube();
    sphere->setPosition(-2.0, 0.0, 0.0);
    cube->setPosition(2.0, 0.0, 0.0);
    this->addChild(sphere);
    this->addChild(cube);
}
```

The code above will produce a composite object containing a sphere and a cube positioned next to each other. The following code shows how to use the *CIvfCubeSphere*. *CIvfComposite* derived classes are very easily implemented. No destruction code is needed since the composite will delete any unreferenced children.

```
#include "IvfCubeSphere.h"
.
.
CIvfCubeSphere* cubeSphere = new CIvfCubeSphere();
.
.
cubeSphere->render();
```

When deriving composite classes it is important to consider how selection should be handled. There are two ways of handling selection in a composite derived class. Children can be selected independently or the entire composite class can be selected. Selection is controlled by the flag UseName. If UseName is set to true the object will render a name in OpenGL. If the children of the composite are to be independently selectable the UseName flag of the composite must be set to false and the UseName flags of the children set to true. Default for composite objects, the UseName flag is set to false. The following code shows the constructors for the two different cases. Independently selectable children:

```
CIvfCubeSphere::CIvfCubeSphere()
    :CIvfComposite()
{
    this->setUseName(false);   // Default behavior

    CIvfSpherePtr sphere = new CIvfSphere();
    CIvfCubePtr cube = new CIvfCube();
    sphere->setPosition(-2.0, 0.0, 0.0);
    sphere->setUseName(true); // Default behavior
    cube->setPosition(2.0, 0.0, 0.0);
    cube->setUseName(true);    // Default behavior
    this->addChild(sphere);
    this->addChild(cube);
}
```

Selectable composite:

```
CIvfCubeSphere::CIvfCubeSphere()
    :CIvfComposite()
{
    this->setUseName(true);

    CIvfSpherePtr sphere = new CIvfSphere();
    CIvfCubePtr cube = new CIvfCube();
    sphere->setPosition(-2.0, 0.0, 0.0);
    sphere->setUseName(false);
    cube->setPosition(2.0, 0.0, 0.0);
    cube->setUseName(false);
    this->addChild(sphere);
    this->addChild(cube);
}
```

9.2 CIvfShape derived classes

CIvfShape is the base of all visible OpenGL classes. A derived *CIvfShape* class must implement the *createGeometry* method. In this method OpenGL drawing code is placed to defined the geometry. The *createSelect* method can be implemented to show an appropriate selection geometry. A simplified version of the Ivf++ cone class is shown to illustrate a *CIvfShape* derived class. Class header:

```
#ifndef _CIvfCone_h_
#define _CIvfCone_h_

#include <ivf/IvfShape.h>

class CIvfCone : public CIvfShape {
private:
    .
    .
public:
    CIvfCone();
    ~CIvfCone();

    .
    .

protected:
    virtual void createSelect();
    virtual void createGeometry();
}

#endif
```

Class implementation:

```
#include "IvfCone.h"

.
.
.

void CIvfCone::createGeometry()
{
    glPushMatrix();
        glPushMatrix();
            glTranslated(0.0,-getHeight()/2.0,0.0);
            glRotated(-90,1.0,0.0,0.0);

            GLUquadricObj* cylinder = gluNewQuadric();
            gluQuadricNormals(cylinder,GLU_SMOOTH);
            gluQuadricTexture(cylinder,GL_TRUE);
            gluQuadricDrawStyle(cylinder,GLU_FILL);
            gluCylinder(cylinder,
                getBottomRadius(),getTopRadius(),
                    getHeight(),
                getSlices(),getStacks());
            gluDeleteQuadric(cylinder);

            GLUquadricObj* bottom = gluNewQuadric();
            gluQuadricNormals(bottom,GLU_SMOOTH);
            gluQuadricTexture(bottom,GL_TRUE);
            gluQuadricDrawStyle(bottom,GLU_FILL);
            gluDisk(bottom,
                0.0, getBottomRadius(),
                getSlices(),1);
            gluDeleteQuadric(bottom);
        glPopMatrix();

        if (getTopRadius()>0.0)
        {
            glPushMatrix();
                glTranslated(0.0,getHeight()/2.0,0.0);
                glRotated(-90,1.0,0.0,0.0);

                GLUquadricObj* top = gluNewQuadric();
                gluQuadricNormals(top,GLU_SMOOTH);
                gluQuadricTexture(top,GL_TRUE);
                gluQuadricDrawStyle(top,GLU_FILL);
                gluDisk(top,
                                0.0, getTopRadius(),
                                getSlices(),1);
                          gluDeleteQuadric(top);
                    glPopMatrix();
              }

        glPopMatrix();
}

.
.

void CIvfCone::createSelect()
{
        m_selectionBox->render();
}
```

9.3 CIvfGLPrimitive derived classes

Advanced geometry can be derived from the *CIvfGLPrimitive* derived classes *CIvfPointSet*, *CIvfLineSet*, *CIvfLineStripSet*, *CIvfTriSet*, *CIvfTriStripSet*, *CIvfQuadSet* and *CIvfQuadStripSet*. These classes implement the different OpenGL drawing primitives. All *CIvfGLPrimitive* derived classes maintains a coordinate set, a color set and a texture coordinate set. Topology is defined by a corresponding set of index set referencing the latter sets. Figure 9.1

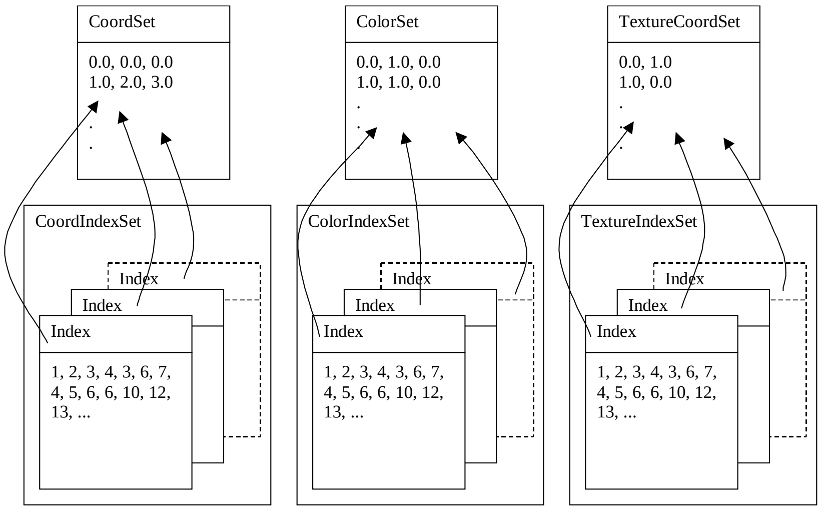


Figure 9.1: *CIvfGLPrimitive* class structure

Below is shown a simple class implementing a 3D plane derived from *CIvfQuad-*

Set. The geometry is defined in the class constructor. Class header:

```
#ifndef _CIvfQuadPlane_h_
#define _CIvfQuadPlane_h_

#include <ivf/IvfQuadSet.h>

class CIvfQuadPlane : public CIvfQuadSet {
private:
    double m_width;
    double m_height;
    double m_ratio;
public:
    CIvfQuadPlane();

    void setWidth(double width);
    void setSize(double width, double height);
    void setTexture(CIvfTexture* texture);
};

#endif
```

Class implementation:

```
#include "IvfQuadPlane.h"

CIvfQuadPlane::CIvfQuadPlane()
    :CIvfQuadSet()
{
    m_width = 1.0;
    m_heigth = 1.0;

    this->addCoord(-width/2.0, -height/2.0, 0.0);
    this->addCoord(width/2.0, -height/2.0, 0.0);
    this->addCoord(width/2.0, height/2.0, 0.0);
    this->addCoord(-width/2.0, height/2.0, 0.0);
    CIvfIndexPtr idx = new CIvfIndex();
    idx->createLinear(4);
    this->addCoordIndex(idx);
    this->addTextureCoord(0.0,0.0);
    this->addTextureCoord(1.0,0.0);
    this->addTextureCoord(1.0,1.0);
    this->addTextureCoord(0.0,1.0);
    idx = new CIvfIndex();
    idx->createLinear(4);
    this->addTextureIndex(idx);
}

void CIvfQuadPlane::setSize(double width, double
    height)
```

```
{
    m_width = width;
    m_height = height;
    m_ratio = width / height;

    this->setCoord(0, -width/2.0, -height/2.0, 0.0);
    this->setCoord(1, width/2.0, -height/2.0, 0.0);
    this->setCoord(2,  width/2.0, height/2.0, 0.0);
    this->setCoord(3, -width/2.0, height/2.0, 0.0);
}

void CIvfQuadPlane::setTexture(CIvfTexture *texture)
{
    int width, height;
    CIvfShape::setTexture(texture);
    texture->getSize(width, height);
    m_ratio = (double)width / (double)height;
    this->setSize(m_width, m_width/m_ratio);
}

void CIvfQuadPlane::setWidth(double width)
{
    this->setSize(width, width/m_ratio);
}
```

9.4 Implementing type information

To implement type information in a derived class the *getClassName* and *isClass* methods have to be implemented. The *getClassName* method should return the name of the derived class. *isClass* should return true if the name passed to the method is the name of the derived class or any of the inherited classes. To make life easier Ivf++ defines a macro, IvfClassInfo for implementing the class. This macro takes two input parameters, a string with the name of the class and the name of the parent class. The code from section 9.1 is modified to contain class information. Class header:

```
#ifndef _CIvfCubeSphere_h_
#define _CIvfCubeSphere_h_

#include <ivf/IvfComposite.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCube.h>

class CIvfCubeSphere : public CIvfComposite {
public:
    CIvfComposite();
    IvfClassInfo("CIvfCubeSphere",CIvfComposite);
};

#endif
```

An alternative to this approach is to use run-time type information (RTTI), which probably is better if supported by the compiler. The class type information in Ivf++ is only provided to give some help when the compiler doesn't support RTTI.

9.5 Declaring a smart pointers

To create a smart pointer for a new class, Ivf++ contains a macro IvfSmartPointer which takes as input the name of the new class. The macro is added before the class declaration. The following example shows how a smart pointer is declared for the CIvfCubeSphere class.

```
#ifndef _CIvfCubeSphere_h_
#define _CIvfCubeSphere_h_

#include <ivf/IvfComposite.h>

IvfSmartPointer(CIvfCubeSphere);

class CIvfCubeSphere : public CIvfComposite {
public:
    CIvfComposite();
};

#endif
```

Chapter 10

Using Application and Class generators

To make it easier to get started with the Ivf++ library special Python based applications are included with the Ivf++ distribution for generating skeleton Ivf++ applications and classes. The application and class generators are available from the Start menu in the Win32 binary installations and as command line tools on both Win32 and Linux/Unix.

Both the application generator and class generator provide both a graphical user interface in addition to a pure command line based interface. In this chapter the graphical user interface versions will be described. The same options are available from the command line version.

10.1 Ivf++ Application Generator

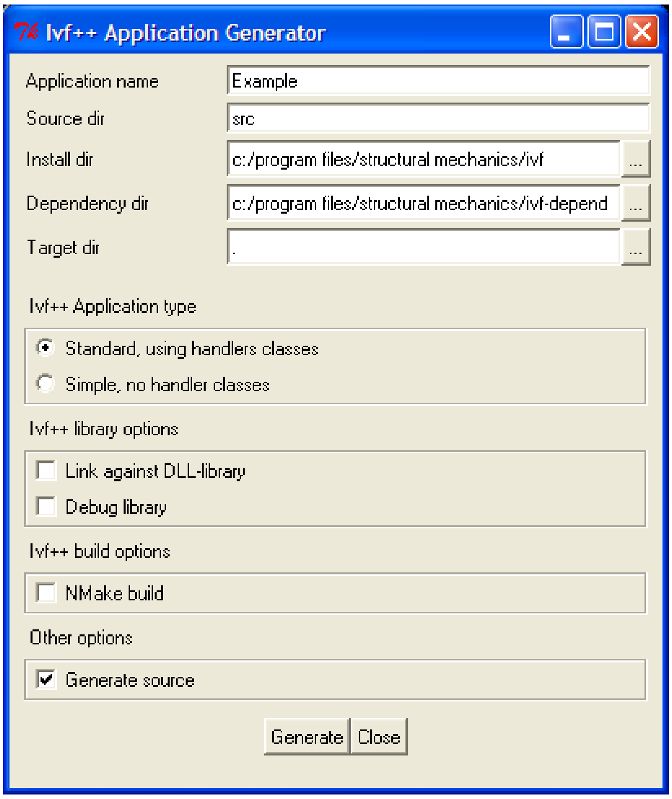
The Ivf++ application generator can generate source and configuration files (CMake) for a skeleton Ivf++ application. The Ivf++ Application generator is started from the command line by issuing the following command:

```
$ ivfappgen gui
```

This brings up the window shown in figure 10.3. The options in the window are described in table 10.1. The build system generated by the application generator requires the CMake tool to be used. To make it easier to use the application generator also generates scripts and conventional Makefiles (Linux/Unix).

To use of the application generator is illustrated by an example. First a directory for the example is created and we set it as the current directory.

```
C:\...\> mkdir testapp
```


Ivf++ Application Generator

Application name: Example
Source dir: src
Install dir: c:/program files/structural mechanics/ivf
Dependency dir: c:/program files/structural mechanics/ivf-depend
Target dir: .

Ivf++ Application type
- [✓ Standard, using handlers classes]
- Simple, no handler classes

Ivf++ library options
- [ ] Link against DLL-library
- [ ] Debug library

Ivf++ build options
- [ ] NMake build

Other options
- [x] Generate source

Figure 10.1: The Ivf++ Application Generator user interface

```
C:\...\> cd testapp
C:\...\testapp>
```

We then start the application generator using the ivfappgen gui command from the command line. In this example, the source code will be located in the same directory as the other build files. The name of the application is changed to "TestApp". We also check the option of using the debug version of the Ivf++ library. The Finished configuration is shown in figure 10.2.

The source file and configuration is written by clicking on the "Generate" button. Figure **??** shows the files generated by the application generator (Win32). The CMakeLists.txt is the main configuration file, which also can be used directly by the CMake tool. The main.cpp is the generated source code for the Ivf++ application. The build_debug.cmd, configure_debug.cmd and clean_debug.cmd are script files for configuring and building the application. The application generator will generate separate script files for each build type, such as build.cmd for building with a static release version of Ivf++, build_shared_debug.cmd for building with a shared debug version of Ivf++. The files generated by each build will also be located in separate

User interface element	description
Application name	Descriptive name of the application. The name will also be used to define application classes and output files.
Source dir	If specified, all generated source files will be placed in a subdirectory with this name. If this is not desired leave the field empty.
Install dir	Ivf++ installation directory.
Dependency dir	Ivf++ dependency installation directory.
Target dir	Directory where the output is generated. Must exist.
NMake build	Build configuration files for building the application using NMake.
Generate source	Enables/disables generation of source. Used if only the build system is to be generated.

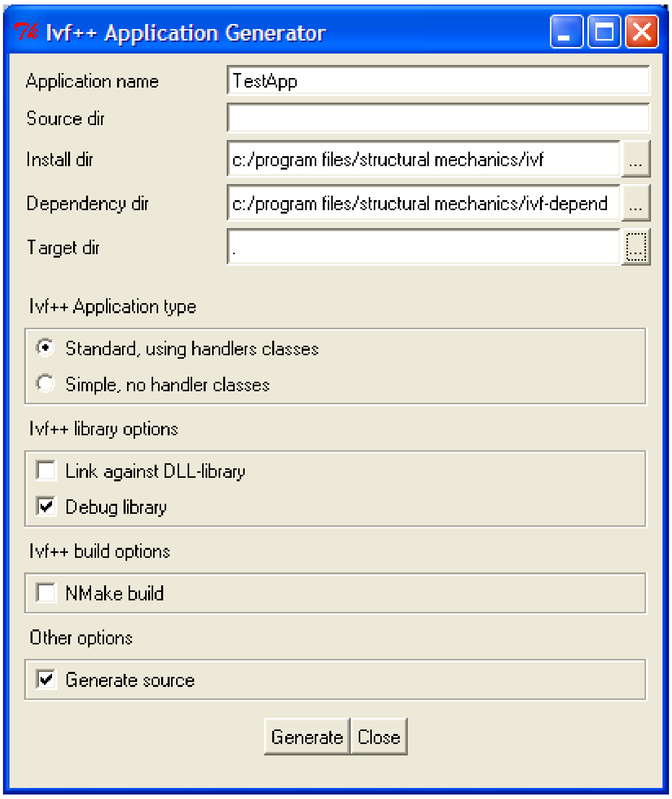
Table 10.1: Handler classes in Ivf++

directories.

The first step in building the generated application is to configure it. In the described example this is done by using the configure_debug command:

```
C:\...\testapp> configure_debug
-- Check for working C compiler: cl
-- Check for working C compiler: cl -- works
-- Check size of void*
-- Check size of void* - done
-- Check for working CXX compiler: cl
-- Check for working CXX compiler: cl -- works
-- Configuring done
-- Generating done
-- Build files have been written to: C:/.../testapp/
   win32_debug_build
```

As the last line suggests the generated build files are located in the C:/.../testapp/win32_debug_bui
directory. The files generated by CMake are Visual Studio solutions files, as shown in figure ??.



Application name: TestApp
Source dir:
Install dir: c:/program files/structural mechanics/ivf
Dependency dir: c:/program files/structural mechanics/ivf-depend
Target dir: .

Ivf++ Application type
- [✓] Standard, using handlers classes
- [ ] Simple, no handler classes

Ivf++ library options
- [ ] Link against DLL-library
- [✓] Debug library

Ivf++ build options
- [ ] NMake build

Other options
- [✓] Generate source

Figure 10.2: Application example parameters

10.2 Ivf++ Class Generator

To make it easier to extend Ivf++ a special Python based class generator is provided with the library. The class generator can be used both from the command line or as a graphical user interface application. The generator can generate the following classes:

- Non-OpenGL, CIvfBase, derived classes.
- OpenGL-enabled, CIvfShape, derived classes.
- CIvfQuadSet, derived classes.
- CIvfTriSet, derived classes.
- CIvfComposite, derived classes.
- Singleton classes.
- Custom classes, not derived from Ivf++.

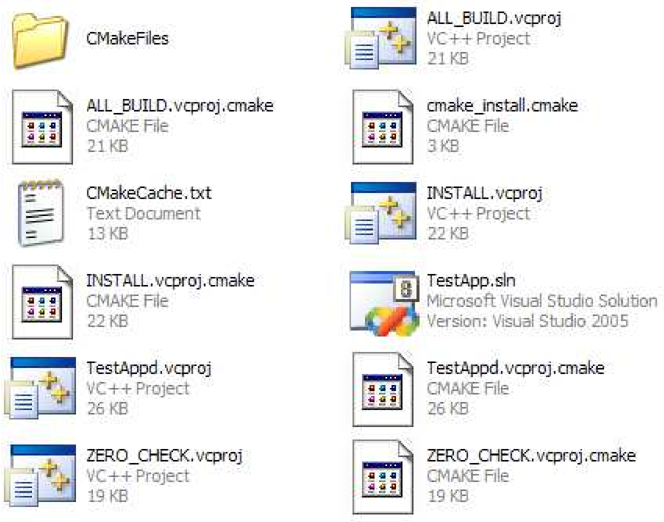


Figure 10.3: Files generated by CMake

10.2.1 Generating classes from the command line

Both Linux and Windows can use the same command lines tools for generating classes. To show the options for the command line tool, execute the ivfclassgen command i a command prompt or terminal. The output of the command is shown below:

```
C:\..>ivfclassgen Ivf Classgen 0.6 - Ivf++ Class
    Template Generator
Copyright (C) 2006 Division of Structural Mechanics
    Usage:

         ivfclassgen classtype|gui [classname] [
             baseclass]

values for classtype are:

         base             CIvfBase derived class
         shape            CIvfShape derived class
         quadset          CIvfQuadSet derived class
         triset           CIvfTriSet derived class
         composite        CIvfComposite derived class
         singleton        singleton class
         custom           Custom derived class

         gui              User interface for generating
             classes
```

To generate a CIvfShape-derived class, CMyClass, the following commands are given:

```
C:\...>ivfclassgen shape MyClass
C:\...>dir

 Volume in drive C is XXXX
 Volume Serial Number is XXXX–XXXX

 Directory of C:\...

2006−07−18  22:27    <DIR>          .
2006−07−18  22:27    <DIR>          ..
2006−07−18  22:27               329 MyClass.cpp
2006−07−18  22:27               276 MyClass.h
               2 File(s)            605 bytes
```

The class generator will automatically add smart pointer declarations, runtime type information, include files and a skeleton implementation. Please note that the class name is prefixed with a ”C”, so the class name is entered as MyClass at the command line. The generated files are not prefixed. The generated, MyClass.h and MyClass.cpp are shown in the following listings:

MyClass.h

```
#ifndef _CMyClass_h_
#define _CMyClass_h_

#include <ivf/IvfShape.h>

IvfSmartPointer(CMyClass);

class CMyClass: public CIvfShape
{
private:

public:
    CMyClass();

    IvfClassInfo("CMyClass", CIvfShape);
protected:
    virtual void createGeometry();
};

#endif
```

MyClass.cpp

```
#include "MyClass.h"

CMyClass::CMyClass() {
    // Add construction code here
}

void CMyClass::createGeometry()
{
    // Rendering code

    glBegin(GL_QUADS);
    glNormal3d( 0.0, 0.0, 1.0);
    glVertex3d( 1.0, 1.0, 0.0);
    glVertex3d(-1.0, 1.0, 0.0);
    glVertex3d(-1.0,-1.0, 0.0);
    glVertex3d( 1.0,-1.0, 0.0);
    glEnd();
}
```

Bibliography

[1] OpenGL, http://www.opengl.org, 2000

[2] M. Kilgard, The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3, http://reality.sgi.com/opengl/spec3/spec3.html, 2000

[3] Lindemann J., Interactive Visualisation Framework - User's guide, Report TVSM-3038, Lund University

[4] B. Spitzak, Fast Light Toolkit FLTK, http://www.fltk.org, 2000

[5] GNU Project, http://www.gnu.org, 2000

[6] The GIMP Toolkit, http://www.gtk.org, 2000

[7] M. Woo et al., OpenGL Programming Guide: the official guide to learning OpenGL, Addison Wesley Longman Inc., 1999

[8] Trolltech AS, http://www.trolltech.com/, 2000

[9] Microsoft Foundation Classes and Templates, http://msdn.microsoft.com/library/devprods/vs6/visualc/vcmfc/vcmfchm.htm, 2000

[10] The GLE Tubing and Extrusion Library, http://linas.org/gle/, 2001

Appendix A

Building and installing

A.1 Building Ivf++ on Linux

To make it easier to build the Ivf++ libraries a new build system has been developed for the 1.0 release. This system automatically handles the download, building and installing dependencies as well as building the main Ivf++ libraries.

A.1.1 Requirements

Before installing the Ivf++ library the following libraries should be installed, however the Ivf++ build system can handle downloading, building and installing these as well.

- OpenGL 1.1 including the GLU library
- The GLE Tubing and Extrusion Library 3.1.x – http://linas.org/gle/
- Fast Light Toolkit 1.1.x – http://www.fltk.org
- FTGL 2.1.2 – http://www.moonlight3D.org/gltt (If building ivffont library)
- Freetype 2.x – http://www.freetype.org (If building ivffont library)
- Recent libpng 1.2.x and libjpeg

The Ivf++ build system also requires the following software packages:

- CMake Cross-platform Make – http://www.cmake.org
- Python 2.3 or higher – http://www.python.org
- Doxygen documentation system – http://www.doxygen.org. For building library reference documentation.

A.1.2 Unpacking the distribution

The first step in the installation, is to unpack the tar-file. This is done with the following commands.

```
$ tar xvzf ivf-1.x.y.tar.gz
```

This will unpack Ivf++ in a directory named ivf-1.x.y

A.1.3 Downloading, building and installing dependencies

The needed dependency packages can be downloaded in the source tree using the ivfbuild command.

```
$ cd ivf-x.x.x
$ ./ivfbuild depends download

Checking build system requirements.

CMake, found.
Doxygen, not found. Documentation generation disabled.


Ivf++ dependency build.

----------------------------------------
Downloading:  gle-3.1.0.tar.gz
----------------------------------------
.
.
.
```

This downloads and patches (see the patches directory form more information) all source packages in the depends-directory.

The packages are configured and build for install in /usr/local using the following command:

```
$ ./ivfbuild depends build /usr/local

Checking build system requirements.

CMake, found.
Doxygen, not found. Documentation generation disabled.
```

```
Ivf++ dependency build.

patching file ./depend/gle-3.1.0/configure
.
.
.
```

When the build has completed it is installed by issuing the following commands as root.

```
# ./ivfbuild depends install
```

This installs all packages needed for building Ivf++ in /usr/local.

A.1.4 Building Ivf++ and installing

When all dependencies has been installed the Ivf++ library can be built. The first step is to configure the the build. This is done using the ivfbuild configure command. Using this command the build type and prefix are set.

```
# ./ivfbuild configure shared_release /usr/local
```

Availble build types are:

shared_release Optimised shared libraries (.so)

shared_debug Shared libraries with debug information (.so)

static_release Optimised static libraries (.a)

static_debug Static libraries with debug information (.a)

The last parameter sets the prefix for the Ivf++ install.

When the configuration is finished a special build library linux2_build is created where the actual build files will be located. No building is done within the src tree. The build is started using the ivfbuild build command.

```
# ./ivfbuild build
```

Ivf++ is then installed using the ivfbuild install command in the directory specified with the prefix option earlier with the configure option.

```
$ su
# ./ivfbuild install
```

To test the installation run any of the example applications installed with the distribution. The example programs can be found in the install directory (/usr/local in the above example) in the /bin directory.

Please note that if you have compiled Ivf++ as shared libraries it is required to either add the /lib directory to the LD_LIBRARY_PATH environment variable or modify the /etc/ld.conf file.

A.2 Building Ivf++ on Windows

This document describes how to build the Windows version of the Ivf++ library from the source tree. A more convenient way of using the library is to use the pre-compiled packages available at sourceforge, which contain binary versions of the Ivf++ library and dependencies.

To make it easier to build the Ivf++ libraries a new build system has been developed for the 1.0 release. This system automatically handles building, downloading and installing dependencies (Linux only) as well as building the main Ivf++ libraries.

A.2.1 Requirements

Before installing the Ivf++ library the following libraries should be installed.

OpenGL 1.1 including the GLU library

The GLE Tubing and Extrusion Library 3.1.x http://linas.org/gle/

Fast Light Toolkit 1.1.x http://www.fltk.org

FTGL 2.1.2 http://www.moonlight3D.org/gltt (If building ivffont library)

Freetype 2.x http://www.freetype.org (If building ivffont library)

Building and installing these libraries on Windows can be quite difficult. To make it more easy to satisfy the above requirement a precompiled package is provided.

The Ivf++ build system also requires the following software packages:

CMake Cross-platform Make http://www.cmake.org

Python 2.3 or higher http://www.python.org

These packages are available as binary installs from the websites above.

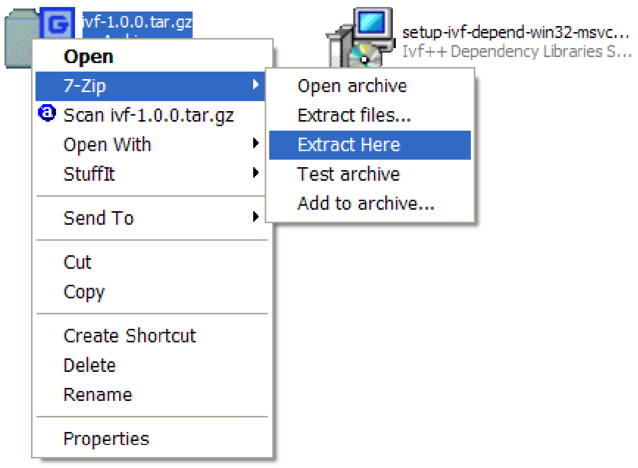


Figure A.1: Unpacking the Ivf++ source package .gz-file

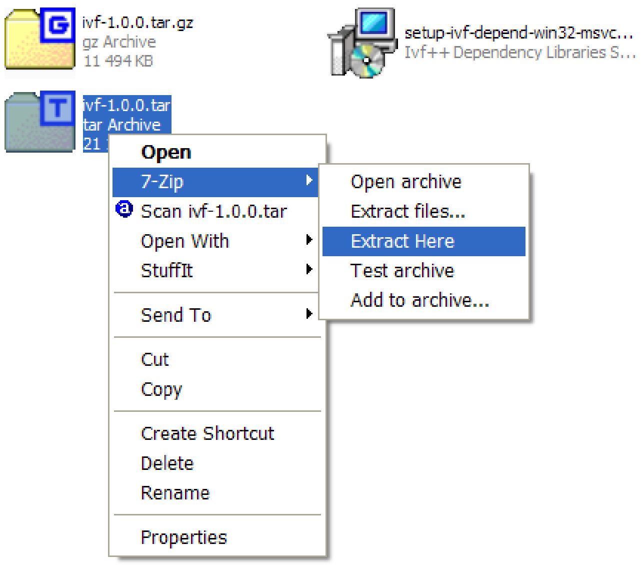
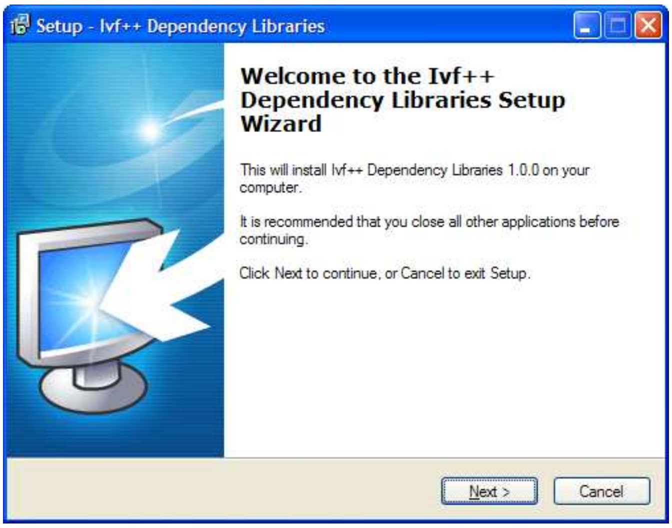


Figure A.2: Unpacking the Ivf++ source package tar archive

A.2.2 Unpacking the distribution

The first step in the installation, is to unpack the source tar-file. Figures A.1, A.2, A.3 and A.4 shows how the source distribution e unpacked using the 7-zip archiver.

When using the binary dependency install it is a good idea to install the package in the same directory where the source package was unpacked. The ivfbuild command will automatically add this library when building Ivf++.


Setup - Ivf++ Dependency Libraries

Welcome to the Ivf++ Dependency Libraries Setup Wizard

This will install Ivf++ Dependency Libraries 1.0.0 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Figure A.3: Installing the Ivf++ dependency package

ivf-depend
ivf-1.0.0

Figure A.4: Ivf++ source and dependency directories ready for building

A.2.3 Configuring the Ivf++ build environment

Before the Ivf++ build system can be used in a Windows environment the build environment must be configured. This is done by modifying the ivfenv.cmd-file. Please change this file to reflect the installation directories of the Python-interpreter and Ivf++ source and dependency directories. A typical file is shown in the following example:

```
@echo off
set PYTHON_ROOT=c:\python24
set IVF_ROOT=d:\dev\ivf
set IVF_DEPEND_ROOT=d:\dev\ivf-depend
set IVF_UTILS=%IVF_ROOT%\utils
```

A.2.4 Building Ivf++

When all dependencies has been installed the Ivf++ library can be built. The first step is to configure the the build. This is done using the ivfbuild configure command. There are two methods of building Ivf++ on Windows. The easiest way is to generate Visual Studio .NET solution files (.sln) that can be built from the IDE. For this build type to work, the include and library paths to the Ivf++ and dependency libraries must be added to the Visual Studio project settings. These settings can be found under **Tools/Options/Projects and Solutions/VC++ directories** in the Visual Studio menus. The second method uses the Microsoft NMake tool to build Ivf++. This builds the library directly from the command line. Configuring Ivf++ to generate solution files is done using the following commands from the command line:

```
C:\..\> ivfbuild configure shared_release
```

A NMake build requires the Visual Studio .NET/2005 build environment to be set using the vsvars.bat batch script.

```
C:\..\> cd "\program files\Microsoft Visual Studio .NET 2003"
C:\..\> cd \Common7\Tools
C:\..\> vsvars
Setting environment forusing Microsoft Visual Studio .NET
2003 tools. (If you have another version of Visual
Studio or Visual C++ installed and wish to use its tools
from the command line, run vcvars32.bat for that version.)
```

A NMake build is then configured with the following command line:

```
C:\..\> ivfbuild configure shared_release nmake
```

```
Checking build system requirements.

CMake, found.
NMake, found. Win32 make file build supported.
Doxygen, found. Documenation generation available.


Configuring Ivf++ build.

-- Check for CL compiler version
-- Check for CL compiler version - 1310
-- Check if this is a free VC compiler
-- Check if this is a free VC compiler - no
-- Check for working C compiler: cl
-- Check for working C compiler: cl -- works
-- Check size of void*
-- Check size of void* - done
-- Check for working CXX compiler: cl
-- Check for working CXX compiler: cl -- works
Ivf++ Windows build
Debug information off
Building shared libraries.
-- Configuring done
-- Generating done
-- Build files have been written to: C:/.../ivf/win32_build

Now run "ivfbuild build", to build Ivf++.
```

Availble build types are:

shared_release Optimised shared libraries (.dll/.lib)

shared_debug Shared libraries with debug information (.dll/.lib)

static_release Optimised static libraries (.lib)

static_debug Static libraries with debug information (.lib)

When the configuration is finished a special build library win32_build is created where the build files are located. No building is done in the src tree. The build is then started using the ivfbuild build (NMake).

```
C:\..\> ivfbuild build

Checking build system requirements.

CMake, found.
NMake, found. Win32 make file build supported.
```

```
Doxygen, found. Documenation generation available.


Building Ivf++.


Microsoft (R) Program Maintenance Utility ...
Copyright (C) Microsoft Corporation.  All ...

.
.
Scanning dependencies of target ivfmath
Building CXX object src/ivfmath/CMakeFiles/ivfmath...
```

A.2.5 Visual C++ 2005 Express edition

Ivf++ can be built successfully using the Visual C++ 2005 Express edition, however it will require some modifications and additions to the Visual Studio application.

The default installation of Visual Studio is mainly targeted towards .NET development and can only build simple console based native Win32 applications. To be able to build window based Win32 application the Microsoft Platform SDK (search microsoft for Windows Server 2003 R2 Platform SDK) is needed and some modifications to the environment, see http://msdn.microsoft.com/vstudio/express/-visualc/usingpsdk

A.2.6 Installing Ivf++ from the binary distribution

The quickest way of using the Ivf++ library on Windows is to use precompiled versions of the library available as windows installers for Visual Studio .NET 2003 and Visual C++ 2005 Express edition, available from SourceForge. When the dependency and library installs have been completed the only thing necessary to get started is to setup the Visual Studio IDE with the location of the Ivf++ include and library files. The following directories should be added.

Include files:

```
[Ivf-install-dir]\include\vc
[Ivf-install-dir]\include
[Ivf-dependency-dir]\include
```

Library files:

```
[Ivf-install-dir]\lib\[static|shared]
[Ivf-dependency-dir]\lib
```

If the automated build tools ”Ivf++ Application Generator” (ivfappgen) or ”Ivf++ Class Generator” (ivfclassgen) is to be used Python 2.4 and CMake are also required.

Appendix B

Tutorial source code

B.1 Using Ivf++ user interface library

```
// ------------------------------------------------------------
//
// Ivf++ colored cube example,
// with interaction, idle processing and timer handling
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfCube.h>
#include <ivf/IvfComposite.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:

    // Camera movement state variables
    int m_beginX;
    int m_beginY;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    // Ivf++ object declarations

    CIvfCamera*    m_camera;
    CIvfCube*      m_cube;
    CIvfLight*     m_light;
    CIvfComposite* m_scene;
public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    // Basic ivfui event methods

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();

    // Mouse event methods

    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);

    // Keyboard event methods

    virtual void onKeyboard(int key, int x, int y);

    // Idle processing

    virtual void onIdle();

    // Timer processing

    virtual bool onTimeout0();
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
```

```
{
    // State variables

    m_angleX = 0.0f;
    m_angleY = 0.0f;
    m_moveX = 0.0f;
    m_moveY = 0.0f;
    m_zoomX = 0.0f;
    m_zoomY = 0.0f;

    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(2.0, 2.0, 2.0);

    // Create a material

    CIvfMaterial* material = new CIvfMaterial();
    material->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    material->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

    // Initialize scene

    m_scene = new CIvfComposite();

    // Create a cube

    m_cube = new CIvfCube();
    m_cube->setMaterial(material);

    m_scene->addChild(m_cube);

    // Create a light

    m_light = new CIvfLight();
    m_light->setPosition(1.0, 1.0, 1.0, 0.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setPerspective(45.0, 0.1, 100.0);
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (this->isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);

        this->redraw();
    }

    if (this->isRightButtonDown())
    {
        if (this->getModifierKey()==CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }
        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);

        this->redraw();
    }
}

// ------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    switch (key) {
    case 'r':
        m_camera->setPosition(2.0, 2.0, 2.0);
        m_camera->setTarget(0.0, 0.0, 0.0);
        this->redraw();
        break;
    case 'i':
        if (this->isIdleProcessing())
            this->disableIdleProcessing();
        else
            this->enableIdleProcessing();
        break;
    case 't':
        if (this->isTimeoutEnabled(0))
            this->disableTimeout(0);
        else
            this->enableTimeout(0.1, 0);
        break;
    default:

        break;
    }
}

// ------------------------------------------------------------
void CExampleWindow::onIdle()
{
    m_camera->rotatePositionY(0.1);
    this->redraw();
}
```

```
// ------------------------------------------------------------
bool CExampleWindow::onTimeout0()
{
    double rx, ry, rz, angle;

    m_cube->getRotationQuat(rx, ry, rz, angle);
    m_cube->setRotationQuat(1.0, 0.0, 0.0, angle+1.0);
    this->redraw();

    return true;
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Cube example");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.2 Placing and rotating objects

```
// ------------------------------------------------------------
//
// Ivf++ Object placement/rotation sample
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfCube.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCylinder.h>
#include <ivf/IvfCone.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:

    // Camera movement state variables

    int m_beginX;
    int m_beginY;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    CIvfCamera*     m_camera;
    CIvfComposite*  m_scene;
    CIvfLight*      m_light;
public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();

    // Mouse event methods

    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // State variables

    m_angleX = 0.0f;
    m_angleY = 0.0f;
    m_moveX = 0.0f;
    m_moveY = 0.0f;
    m_zoomX = 0.0f;
    m_zoomY = 0.0f;

    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 4.0, 9.0);
    m_camera->setPerspective(45.0, 0.1, 100.0);

    // Create a materials

    CIvfMaterial* redMaterial = new CIvfMaterial();
    redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    redMaterial->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

    CIvfMaterial* greenMaterial = new CIvfMaterial();
    greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    greenMaterial->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);

    CIvfMaterial* blueMaterial = new CIvfMaterial();
    blueMaterial->setDiffuseColor(0.0f, 0.0f, 1.0f, 1.0f);
    blueMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    blueMaterial->setAmbientColor(0.0f, 0.0f, 0.5f, 1.0f);

    CIvfMaterial* yellowMaterial = new CIvfMaterial();
    yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f);
    yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f);

    // Create scene composite

    m_scene = new CIvfComposite();

    // Create objects

    CIvfCube* cube = new CIvfCube();
    cube->setMaterial(redMaterial);
    cube->setPosition(2.0, 0.0, 2.0);
    m_scene->addChild(cube);

    CIvfSphere* sphere = new CIvfSphere();
    sphere->setMaterial(greenMaterial);
    sphere->setPosition(-2.0, 0.0, 2.0);
    m_scene->addChild(sphere);

    CIvfCylinder* cylinder = new CIvfCylinder();
    cylinder->setMaterial(blueMaterial);
    cylinder->setPosition(-2.0, 0.0, -2.0);
    m_scene->addChild(cylinder);

    CIvfCone* cone = new CIvfCone();
    cone->setMaterial(yellowMaterial);
    cone->setPosition(2.0, 0.0, -2.0);
    cone->setRotationQuat(0.0, 0.0, 1.0, 45.0);
    m_scene->addChild(cone);

    CIvfAxis* axis = new CIvfAxis();
    m_scene->addChild(axis);

    // Create a light

    m_light = new CIvfLight();
    m_light->setPosition(1.0, 1.0, 1.0, 0.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}
```

```
// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (this->isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);

        this->redraw();
    }

    if (this->isRightButtonDown())
    {
        if (this->getModifierKey()==CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }

        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);

        this->redraw();
    }
}

// ------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Object placement example");
    window->show();

    // Enter main application loop
    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.3 Creating a movable robot arm

```
// ------------------------------------------------------------
//
// Ivf++ Robot arm example with movable camera
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCylinder.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:

    // Camera movement state variables

    int m_beginX;
    int m_beginY;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    CIvfCamera*     m_camera;
    CIvfComposite*  m_scene;
    CIvfLight*      m_light;

    // Robot state variables

    double m_alfa;
    double m_beta;
    double m_gamma;
    double m_delta;

    // Robot arm transforms

    CIvfTransform*  m_part1;
    CIvfTransform*  m_part2;
    CIvfTransform*  m_part3;
    CIvfTransform*  m_arm;

    // Routine for updating the arm

    void updateArm();
public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};


    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onKeyboard(int key, int x, int y);
    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // Initialize variables

    m_alfa = 0.0;
    m_beta = -45.0;
    m_gamma = 90.0;
    m_delta = 75.0;

    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 5.0, 5.0);
    m_camera->setPerspective(45.0, 0.1, 100.0);

    // Create a materials

    CIvfMaterial* redMaterial = new CIvfMaterial();
    redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    redMaterial->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

    CIvfMaterial* greenMaterial = new CIvfMaterial();
    greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    greenMaterial->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);

    CIvfMaterial* blueMaterial = new CIvfMaterial();
    blueMaterial->setDiffuseColor(0.0f, 0.0f, 1.0f, 1.0f);
    blueMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    blueMaterial->setAmbientColor(0.0f, 0.0f, 0.5f, 1.0f);

    CIvfMaterial* yellowMaterial = new CIvfMaterial();
    yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f);
    yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f);

    // Create scene composite

    m_scene = new CIvfComposite();

    // Create robot base

    CIvfCylinder* lowerBase = new CIvfCylinder();
    lowerBase->setHeight(0.2);
    lowerBase->setMaterial(yellowMaterial);
    m_scene->addChild(lowerBase);

    CIvfCylinder* upperBase = new CIvfCylinder();
```

```
upperBase->setHeight(0.3);
upperBase->setRadius(0.5);
upperBase->setPosition(0.0, 0.1 + 0.15, 0.0);
upperBase->setMaterial(blueMaterial);
m_scene->addChild(upperBase);

// Create part1

m_part1 = new CIvfTransform();

CIvfSphere* sphere = new CIvfSphere();
sphere->setRadius(0.1);
sphere->setMaterial(redMaterial);
m_part1->addChild(sphere);

CIvfCylinder* cylinder = new CIvfCylinder();
cylinder->setRadius(0.1);
cylinder->setHeight(1.0);
cylinder->setMaterial(greenMaterial);
cylinder->setPosition(0.0, 0.1 + 0.5, 0.0);

m_part1->addChild(cylinder);
m_part1->setPosition(0.0, 1.2, 0.0);
m_part1->setRotationQuat(0.0, 0.0, 1.0, m_delta);

// Create part2

m_part2 = new CIvfTransform();

sphere = new CIvfSphere();
sphere->setRadius(0.1);
sphere->setMaterial(redMaterial);
m_part2->addChild(sphere);

cylinder = new CIvfCylinder();
cylinder->setRadius(0.1);
cylinder->setHeight(1.0);
cylinder->setMaterial(greenMaterial);
cylinder->setPosition(0.0, 0.1 + 0.5, 0.0);
m_part2->addChild(cylinder);
m_part2->addChild(m_part1);
m_part2->setPosition(0.0, 1.2, 0.0);
m_part2->setRotationQuat(0.0, 0.0, 1.0, m_gamma);

// Create part3

m_part3 = new CIvfTransform();

sphere = new CIvfSphere();
sphere->setRadius(0.1);
sphere->setMaterial(redMaterial);
```

```
    m_part3->addChild(sphere);

    cylinder = new CIvfCylinder();
    cylinder->setRadius(0.1);
    cylinder->setHeight(1.0);
    cylinder->setMaterial(greenMaterial);
    cylinder->setPosition(0.0, 0.1 + 0.5, 0.0);
    m_part3->addChild(cylinder);
    m_part3->addChild(m_part2);
    m_part3->setPosition(0.0, 0.0, 0.0);
    m_part3->setRotationQuat(0.0, 0.0, 1.0, m_beta);

    // Create complete arm

    m_arm = new CIvfTransform();
    m_arm->addChild(m_part3);
    m_arm->setPosition(0.0, 0.1+0.4, 0.0);
    m_arm->setRotationQuat(0.0, 1.0, 0.0, m_alfa);

    m_scene->addChild(m_arm);

    CIvfAxis* axis = new CIvfAxis();
    axis->setSize(1.5);
    m_scene->addChild(axis);

    // Create a light

    m_light = new CIvfLight();
    m_light->setPosition(1.0, 1.0, 1.0, 0.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}

// ------------------------------------------------------------
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    switch (key) {
    case 'a':
        m_alfa += 5.0;
        updateArm();
        break;
    case 'z':
        m_alfa -= 5.0;
        updateArm();
        break;
    case 's':
        m_beta += 5.0;
        updateArm();
        break;
    case 'x':
        m_beta -= 5.0;
        updateArm();
        break;
    case 'd':
        m_gamma += 5.0;
        updateArm();
        break;
    case 'c':
        m_gamma -= 5.0;
        updateArm();
        break;
    case 'f':
        m_delta += 5.0;
        updateArm();
        break;
    case 'v':
        m_delta -= 5.0;
        updateArm();
        break;
    default:
        break;
    }
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}
```

```
// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);

        redraw();
    }

    if (isRightButtonDown())
    {
        if (getModifierKey() == CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }
        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);

        redraw();
    }
}

// ------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
void CExampleWindow::updateArm()
{
    m_arm->setRotationQuat(0.0, 1.0, 0.0, m_alfa);
    m_part1->setRotationQuat(0.0, 0.0, 1.0, m_delta);
    m_part2->setRotationQuat(0.0, 0.0, 1.0, m_gamma);
    m_part3->setRotationQuat(0.0, 0.0, 1.0, m_beta);

    redraw();
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Robot example");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.4 Scene lighting

```
// ------------------------------------------------------------
//
// Ivf++ lighting example
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfLightModel.h>
#include <ivf/IvfMesh.h>
#include <ivf/IvfLightingState.h>

#include <ivfmath/IvfVec3d.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:

    // Camera movement state variables

    int m_beginX;
    int m_beginY;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    CIvfCamera*     m_camera;
    CIvfComposite*  m_scene;
    CIvfLight*      m_light;
    CIvfLightModel* m_lightModel;
    CIvfSphere*     m_lightSphere;

    CIvfVec3d       m_direction;
    double          m_speed;
public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();

    // Mouse event methods

    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);

    // Keyboard events

    virtual void onKeyboard(int key, int x, int y);

    // Timer events

    virtual bool onTimeout0();
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // State variables

    m_angleX = 0.0f;
    m_angleY = 0.0f;
    m_moveX = 0.0f;
    m_moveY = 0.0f;
    m_zoomX = 0.0f;
    m_zoomY = 0.0f;

    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 0.0, 9.0);
    m_camera->setPerspective(60.0, 0.1, 40.0);

    // Create a materials
    CIvfMaterial* redMaterial = new CIvfMaterial();
    redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    redMaterial->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

    CIvfMaterial* greenMaterial = new CIvfMaterial();
    greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    greenMaterial->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);

    CIvfMaterial* blueMaterial = new CIvfMaterial();
    blueMaterial->setDiffuseColor(0.0f, 0.0f, 1.0f, 1.0f);
    blueMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    blueMaterial->setAmbientColor(0.0f, 0.0f, 0.5f, 1.0f);

    CIvfMaterial* yellowMaterial = new CIvfMaterial();
    yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f);
    yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f);

    CIvfMaterial* whiteMaterial = new CIvfMaterial();
    whiteMaterial->setDiffuseColor(1.0f, 1.0f, 1.0f, 1.0f);
    whiteMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    whiteMaterial->setAmbientColor(0.5f, 0.5f, 0.5f, 1.0f);

    // Create scene composite

    m_scene = new CIvfComposite();

    // Create a sphere representing the light

    CIvfLightingState* lightState = new CIvfLightingState();
    lightState->setLighting(false);

    m_lightSphere = new CIvfSphere();
    m_lightSphere->setMaterial(whiteMaterial);
    m_lightSphere->setRadius(0.1);
    m_lightSphere->setRenderState(lightState);
    m_lightSphere->setPosition(-2.0, 0.0, 2.0);
    m_scene->addChild(m_lightSphere);

    // Create a sphere in the middle

    CIvfSphere* sphere = new CIvfSphere();
    sphere->setRadius(1.0);
    sphere->setMaterial(redMaterial);
    sphere->setStacks(12);
    sphere->setSlices(20);
    m_scene->addChild(sphere);
```

```
// Create a room

CIvfMesh* floor = new CIvfMesh();
floor->setMeshType(CIvfMesh::MT_ORDER_2);
floor->setMeshResolution(30,30);
floor->createMesh(10.0, 10.0);
floor->setMaterial(redMaterial);
floor->setPosition(0.0, -3.0, 0.0);

m_scene->addChild(floor);

CIvfMesh* roof = new CIvfMesh();
roof->setMeshType(CIvfMesh::MT_ORDER_2);
roof->setMeshResolution(30,30);
roof->createMesh(10.0, 10.0);
roof->setMaterial(greenMaterial);
roof->setPosition(0.0, 3.0, 0.0);
roof->setRotationQuat(0.0, 0.0, 1.0, 180.0f);
m_scene->addChild(roof);

CIvfMesh* wall1 = new CIvfMesh();
wall1->setMeshType(CIvfMesh::MT_ORDER_2);
wall1->setMeshResolution(30,30);
wall1->createMesh(10.0, 6.0);
wall1->setMaterial(blueMaterial);
wall1->setPosition(0.0, 0.0, -5.0);
wall1->setRotationQuat(1.0, 0.0, 0.0, 90.0f);
m_scene->addChild(wall1);

CIvfMesh* wall2 = new CIvfMesh();
wall2->setMeshType(CIvfMesh::MT_ORDER_2);
wall2->setMeshResolution(30,30);
wall2->createMesh(10.0, 6.0);
wall2->setMaterial(blueMaterial);
wall2->setPosition(0.0, 0.0, 5.0);
wall2->setRotationQuat(1.0, 0.0, 0.0, -90.0f);
m_scene->addChild(wall2);

CIvfMesh* wall3 = new CIvfMesh();
wall3->setMeshType(CIvfMesh::MT_ORDER_2);
wall3->setMeshResolution(30,30);
wall3->createMesh(6.0, 10.0);
wall3->setMaterial(yellowMaterial);
wall3->setPosition(5.0, 0.0, 0.0);
wall3->setRotationQuat(0.0, 0.0, 1.0, 90.0f);
m_scene->addChild(wall3);

CIvfMesh* wall4 = new CIvfMesh();
wall4->setMeshType(CIvfMesh::MT_ORDER_2);
wall4->setMeshResolution(30,30);
wall4->createMesh(6.0, 10.0);
```

```
    wall4->setMaterial(yellowMaterial);
    wall4->setPosition(-5.0, 0.0, 0.0);
    wall4->setRotationQuat(0.0, 0.0, 1.0, -90.0f);
    m_scene->addChild(wall4);

    // Create a light

    m_light = new CIvfLight();
    m_light->setType(CIvfLight::LT_POINT);
    m_light->setPosition(-2.0, 0.0, 2.0);
    m_light->setSpotCutoff(70.0f);
    m_light->setSpotExponent(20.0f);
    m_light->setDiffuse(1.0f, 1.0f, 1.0f, 1.0f);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);

    m_lightModel = new CIvfLightModel();
    m_lightModel->addLight(m_light);

    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);

    // Setup moving light

    m_direction.setComponents(1.0, 1.0, 1.0);
    m_speed = 0.06;

    enableTimeout(0.01, 0);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_camera->render();
    m_lightModel->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}
```

```
// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (this->isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);

        this->redraw();
    }

    if (this->isRightButtonDown())
    {
        if (this->getModifierKey()==CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }

        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);
        this->redraw();
    }
}

// ---------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ---------------------------------------------------------------
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    float angle;
    float exponent;

    switch (key) {
    case 'l':
        if (m_light->getType()==CIvfLight::LT_POINT)
            m_light->setType(CIvfLight::LT_SPOT);
        else
            m_light->setType(CIvfLight::LT_POINT);
        break;
    case 'a':
        angle = m_light->getSpotCutoff();
        angle += 5.0f;
        cout << "Angle = " << angle  << endl;
        m_light->setSpotCutoff(angle);
        break;
    case 'z':
        angle = m_light->getSpotCutoff();
        angle -= 5.0f;
        cout << "Angle = " << angle  << endl;
        m_light->setSpotCutoff(angle);
        break;
    case 's':
        exponent = m_light->getSpotExponent();
        exponent += 1.0f;
        cout << "Exponent = " << exponent  << endl;
        m_light->setSpotExponent(exponent);
        break;
    case 'x':
        exponent = m_light->getSpotExponent();
        exponent -= 1.0f;
        cout << "Exponent = " << exponent  << endl;
        m_light->setSpotExponent(exponent);
        break;
    case 'd':
        m_speed += 0.01;
        break;
    case 'c':
        m_speed -= 0.01;
        break;
    default:
        break;
    }
}

// ------------------------------------------------------------
bool CExampleWindow::onTimeout0()
{
    CIvfVec3d pos;
    double x, y, z;
    double ex, ey, ez;

    pos = m_lightSphere->getPosition();
    pos = pos + m_direction * m_speed;

    pos.getComponents(x, y, z);
    m_direction.getComponents(ex, ey, ez);

    if ((x>4.8)||(x<-4.8))
        ex = -ex;

    if ((y>2.8)||(y<-2.8))
        ey = -ey;

    if ((z>4.8)||(z<-4.8))
        ez = -ez;

    m_direction.setComponents(ex, ey, ez);

    m_lightSphere->setPosition(pos);
    pos.getComponents(x, y, z);
    m_light->setPosition(x, y, z);
    m_light->setSpotDirection(ex, ey, ez);

    this->redraw();

    return true;
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

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

```
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Lighting example");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.5 Textures

```
// ------------------------------------------------------------
//
// Ivf++ Texture mapping example
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfTexture.h>
#include <ivf/IvfQuadPlane.h>
```

```
#include <ivfimage/IvfPngImage.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:
    CIvfCamera*     m_camera;
    CIvfComposite*  m_scene;
    CIvfLight*      m_light;

    CIvfTexture*    m_logoTexture;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    int m_beginX;
    int m_beginY;

public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
    virtual void onKeyboard(int key, int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // Initialize variables

    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 0.0, 6.0);

    // Create a materials

    CIvfMaterial* material = new CIvfMaterial();
    material->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    material->setAmbientColor(0.3f, 0.3f, 0.3f, 1.0f);

    // Create scene composite

    m_scene = new CIvfComposite();

    // Create images

    CIvfPngImage* logoImage = new CIvfPngImage();
    logoImage->setFileName("images/ivf.png");
    logoImage->read();

    // Create a texture

    m_logoTexture = new CIvfTexture();
    m_logoTexture->setImage(logoImage);
    m_logoTexture->setFilters(GL_NEAREST, GL_NEAREST);
    m_logoTexture->setGenerateMipmaps(true);

    // Create a quad plane

    CIvfQuadPlane* logo = new CIvfQuadPlane();
    logo->setSize(1.8, 1.8);;
    logo->setMaterial(material);
    logo->setTexture(m_logoTexture);

    m_scene->addChild(logo);

    // Create a light

    m_light = new CIvfLight();
    m_light->setType(CIvfLight::LT_DIRECTIONAL);
    m_light->setDirection(1.0, 1.0, 1.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
```

```
{
    m_camera->setPerspective(45.0, 0.1, 100.0);
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}

// ------------------------------------------------------------
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    switch (key) {
    case '1':
        m_logoTexture->setFilters(GL_NEAREST, GL_NEAREST);
        break;
    case '2':
        m_logoTexture->setFilters(GL_LINEAR,  GL_LINEAR);
        break;
    case '3':
        m_logoTexture->setFilters(GL_LINEAR_MIPMAP_LINEAR, GL_LINEAR);
        break;
    case 'd':
        m_logoTexture->setMode(GL_DECAL);
        break;
    case 'b':
        m_logoTexture->setMode(GL_BLEND);
        break;
    case 'm':
        m_logoTexture->setMode(GL_MODULATE);
        break;
    default:

        break;
    }

    m_logoTexture->bind();
    redraw();
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);

        redraw();
    }

    if (isRightButtonDown())
    {
        if (getModifierKey() == CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }
        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);
        redraw();
    }
}

// ------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Texture mapping example");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.6 Advanced geometry

```
// ------------------------------------------------------------
```

```
//
// Ivf++ Advanced geometry example
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCylinder.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfTexture.h>

// CIvfGLPrimitive derived classes

#include <ivf/IvfPointSet.h>
#include <ivf/IvfLineSet.h>
#include <ivf/IvfLineStripSet.h>
#include <ivf/IvfTriSet.h>
#include <ivf/IvfTriStripSet.h>
#include <ivf/IvfQuadSet.h>

// From the image library

#include <ivfimage/IvfPngImage.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:
    CIvfCamera*     m_camera;
    CIvfComposite*  m_scene;
    CIvfLight*      m_light;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    int m_beginX;
    int m_beginY;

public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // Initialize variables

    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(-6.6, 9.2, 10.5);
    m_camera->setTarget(1.1, 0.1, 0.6);

    // Create a materials

    CIvfMaterial* greenMaterial = new CIvfMaterial();
    greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    greenMaterial->setAmbientColor(0.0f, 0.2f, 0.0f, 1.0f);

    CIvfMaterial* redMaterial = new CIvfMaterial();
    redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    redMaterial->setAmbientColor(0.2f, 0.0f, 0.0f, 1.0f);

    // Create textures

    CIvfPngImage* logoImage = new CIvfPngImage();
    logoImage->setFileName("images/ivf.png");
    logoImage->read();

    CIvfTexture* logoTexture = new CIvfTexture();
    logoTexture->setImage(logoImage);

    // Create scene composite

    m_scene = new CIvfComposite();

    // Create a PointSet

    CIvfPointSet* pointSet = new CIvfPointSet();

    pointSet->addCoord(0.0, 0.0, 0.0);
    pointSet->addCoord(0.0, 0.0, 1.0);
    pointSet->addCoord(1.0, 0.5, 0.0);
    pointSet->addCoord(1.0, 0.5, 1.0);
    pointSet->addCoord(2.0, 0.5, 0.0);
    pointSet->addCoord(2.0, 0.5, 1.0);
    pointSet->addCoord(3.0, 0.0, 0.0);
    pointSet->addCoord(3.0, 0.0, 1.0);

    CIvfIndex* coordIdx = new CIvfIndex();
    coordIdx->createLinear(8);

    pointSet->addCoordIndex(coordIdx);

    pointSet->addColor(0.0, 0.0, 1.0);
    pointSet->addColor(0.0, 1.0, 0.0);
    pointSet->addColor(0.0, 1.0, 1.0);
    pointSet->addColor(1.0, 0.0, 0.0);
    pointSet->addColor(1.0, 0.0, 1.0);
    pointSet->addColor(1.0, 1.0, 0.0);
    pointSet->addColor(1.0, 1.0, 1.0);
    pointSet->addColor(0.0, 0.0, 1.0);

    CIvfIndex* colorIdx = new CIvfIndex();
    colorIdx->createLinear(8);
    pointSet->addColorIndex(colorIdx);
    pointSet->setUseColor(true);

    pointSet->setPosition(-3.0, 0.0, 3.0);
    pointSet->setPointSize(3);

    m_scene->addChild(pointSet);
```

```
// Create a LineSet

CIvfLineSet* lineSet = new CIvfLineSet();

lineSet->addCoord(0.0, 0.0, 0.0);
lineSet->addCoord(0.0, 0.0, 1.0);
lineSet->addCoord(1.0, 0.5, 0.0);
lineSet->addCoord(1.0, 0.5, 1.0);
lineSet->addCoord(2.0, 0.5, 0.0);
lineSet->addCoord(2.0, 0.5, 1.0);
lineSet->addCoord(3.0, 0.0, 0.0);
lineSet->addCoord(3.0, 0.0, 1.0);

coordIdx = new CIvfIndex();
coordIdx->createLinear(8);

lineSet->addCoordIndex(coordIdx);

lineSet->addColor(0.0, 0.0, 1.0);
lineSet->addColor(0.0, 1.0, 0.0);
lineSet->addColor(0.0, 1.0, 1.0);
lineSet->addColor(1.0, 0.0, 0.0);
lineSet->addColor(1.0, 0.0, 1.0);
lineSet->addColor(1.0, 1.0, 0.0);
lineSet->addColor(1.0, 1.0, 1.0);
lineSet->addColor(0.0, 0.0, 1.0);

colorIdx = new CIvfIndex();
colorIdx->createLinear(8);

lineSet->addColorIndex(colorIdx);

lineSet->setPosition(1.5, 0.0, 3.0);
lineSet->setMaterial(redMaterial);
lineSet->setUseColor(true);
lineSet->setLineWidth(2);

m_scene->addChild(lineSet);

// Create a LineStripSet

CIvfLineStripSet* lineStripSet = new CIvfLineStripSet();

lineStripSet->addCoord(0.0, 0.0, 0.0);
lineStripSet->addCoord(0.0, 0.0, 1.0);
lineStripSet->addCoord(1.0, 0.5, 0.0);
lineStripSet->addCoord(1.0, 0.5, 1.0);
lineStripSet->addCoord(2.0, 0.5, 0.0);
lineStripSet->addCoord(2.0, 0.5, 1.0);
lineStripSet->addCoord(3.0, 0.0, 0.0);
```

```
    lineStripSet->addCoord(3.0, 0.0, 1.0);

    coordIdx = new CIvfIndex();
    coordIdx->createLinear(8);

    lineStripSet->addCoordIndex(coordIdx);

    lineStripSet->addColor(0.0, 0.0, 1.0);
    lineStripSet->addColor(0.0, 1.0, 0.0);
    lineStripSet->addColor(0.0, 1.0, 1.0);
    lineStripSet->addColor(1.0, 0.0, 0.0);
    lineStripSet->addColor(1.0, 0.0, 1.0);
    lineStripSet->addColor(1.0, 1.0, 0.0);
    lineStripSet->addColor(1.0, 1.0, 1.0);
    lineStripSet->addColor(0.0, 0.0, 1.0);

    colorIdx = new CIvfIndex();
    colorIdx->createLinear(8);

    lineStripSet->addColorIndex(colorIdx);

    lineStripSet->setPosition(1.5, 3.0, 3.0);
    lineStripSet->setUseColor(true);
    lineStripSet->setLineWidth(2);

    m_scene->addChild(lineStripSet);

    // Create a TriSet

    CIvfTriSet* triSet = new CIvfTriSet();

    triSet->addCoord(0.0,0.0,2.0);
    triSet->addCoord(1.0,0.3,2.0);
    triSet->addCoord(2.0,0.0,2.0);
    triSet->addCoord(0.0,0.3,1.0);
    triSet->addCoord(1.0,0.5,1.0);
    triSet->addCoord(2.0,0.3,1.0);
    triSet->addCoord(0.0,0.0,0.0);
    triSet->addCoord(1.0,0.3,0.0);
    triSet->addCoord(2.0,0.0,0.0);

    coordIdx = new CIvfIndex();
    coordIdx->add(0,1,4);
    coordIdx->add(0,4,3);
    coordIdx->add(1,2,5);
    coordIdx->add(1,5,4);
    coordIdx->add(3,4,7);
    coordIdx->add(3,7,6);
    coordIdx->add(4,5,8);
    coordIdx->add(4,8,7);
```

```
triSet->addCoordIndex(coordIdx);

triSet->addTextureCoord(0.0,0.0);
triSet->addTextureCoord(0.5,0.0);
triSet->addTextureCoord(1.0,0.0);
triSet->addTextureCoord(0.0,0.5);
triSet->addTextureCoord(0.5,0.5);
triSet->addTextureCoord(1.0,0.5);
triSet->addTextureCoord(0.0,1.0);
triSet->addTextureCoord(0.5,1.0);
triSet->addTextureCoord(1.0,1.0);

CIvfIndex* textureIdx = new CIvfIndex();
textureIdx->assignFrom(coordIdx);

triSet->addTextureIndex(textureIdx);

triSet->setMaterial(greenMaterial);
triSet->setTexture(logoTexture);
//triSet->setUseVertexNormals(true);
triSet->setPosition(-3.0, 0.0, -3.0);

m_scene->addChild(triSet);

// Create a TriStripSet

CIvfTriStripSet* triStripSet = new CIvfTriStripSet();

triStripSet->addCoord(0.0, 0.0, 0.0);
triStripSet->addCoord(0.0, 0.0, 1.0);
triStripSet->addCoord(1.0, 0.5, 0.0);
triStripSet->addCoord(1.0, 0.5, 1.0);
triStripSet->addCoord(2.0, 0.5, 0.0);
triStripSet->addCoord(2.0, 0.5, 1.0);
triStripSet->addCoord(3.0, 0.0, 0.0);
triStripSet->addCoord(3.0, 0.0, 1.0);

triStripSet->addCoord(0.0, 1.0, 0.0);
triStripSet->addCoord(0.0, 1.0, 1.0);
triStripSet->addCoord(1.0, 1.5, 0.0);
triStripSet->addCoord(1.0, 1.5, 1.0);
triStripSet->addCoord(2.0, 1.5, 0.0);
triStripSet->addCoord(2.0, 1.5, 1.0);
triStripSet->addCoord(3.0, 1.0, 0.0);
triStripSet->addCoord(3.0, 1.0, 1.0);

coordIdx = new CIvfIndex();
coordIdx->createLinear(8);

triStripSet->addCoordIndex(coordIdx);
```

```
    coordIdx = new CIvfIndex();
    coordIdx->createLinear(8,8);

    triStripSet->addCoordIndex(coordIdx);

    triStripSet->addColor(0.0, 0.0, 1.0);
    triStripSet->addColor(0.0, 1.0, 0.0);
    triStripSet->addColor(0.0, 1.0, 1.0);
    triStripSet->addColor(1.0, 0.0, 0.0);
    triStripSet->addColor(1.0, 0.0, 1.0);
    triStripSet->addColor(1.0, 1.0, 0.0);
    triStripSet->addColor(1.0, 1.0, 1.0);
    triStripSet->addColor(0.0, 0.0, 1.0);

    colorIdx = new CIvfIndex();
    colorIdx->createLinear(8);

    triStripSet->addColorIndex(colorIdx);

    triStripSet->setMaterial(redMaterial);
    triStripSet->setUseColor(true);
    triStripSet->setUseVertexNormals(true);
    triStripSet->setPosition(1.5, 0.0, -3.0);

    m_scene->addChild(triStripSet);

    // Create a QuadSet

    CIvfQuadSet* quadSet = new CIvfQuadSet();

    quadSet->addCoord(0.0,0.0,1.0);
    quadSet->addCoord(1.0,0.0,1.0);
    quadSet->addCoord(1.0,0.0,0.0);
    quadSet->addCoord(0.0,0.0,0.0);
    quadSet->addCoord(0.0,1.0,1.0);
    quadSet->addCoord(1.0,1.0,1.0);
    quadSet->addCoord(1.0,1.0,0.0);
    quadSet->addCoord(0.0,1.0,0.0);

    coordIdx = new CIvfIndex();
    coordIdx->add(0,1,5,4);
    coordIdx->add(1,2,6,5);
    coordIdx->add(2,3,7,6);
    coordIdx->add(3,0,4,7);
    coordIdx->add(4,5,6,7);
    coordIdx->add(0,3,2,1);

    quadSet->addCoordIndex(coordIdx);

    quadSet->addColor(0.0, 0.0, 1.0);
    quadSet->addColor(0.0, 1.0, 0.0);
    quadSet->addColor(0.0, 1.0, 1.0);
    quadSet->addColor(1.0, 0.0, 0.0);
    quadSet->addColor(1.0, 0.0, 1.0);
    quadSet->addColor(1.0, 1.0, 0.0);
    quadSet->addColor(1.0, 1.0, 1.0);
    quadSet->addColor(0.0, 0.0, 1.0);

    colorIdx = new CIvfIndex();
    colorIdx->assignFrom(coordIdx);

    quadSet->addColorIndex(colorIdx);

    quadSet->setUseColor(true);
    quadSet->setPosition(-3.0, 3.0, -3.0);

    m_scene->addChild(quadSet);

    CIvfAxis* axis = new CIvfAxis();
    axis->setSize(1.5);
    m_scene->addChild(axis);

    // Create a light

    m_light = new CIvfLight();
    m_light->setPosition(1.0, 1.0, 1.0, 0.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setPerspective(45.0, 0.1, 100.0);
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}
```

```
// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;
        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);
        redraw();
    }

    if (isRightButtonDown())
    {
        if (getModifierKey() == CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }
        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);

        redraw();
    }
}
```

```
// ------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Advanced geometry");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.7 Extrusions

```
// ------------------------------------------------------------
//
// Ivf++ Extrusion example
//
// ------------------------------------------------------------
```

```
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfExtrusion.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:
    CIvfCamera*     m_camera;
    CIvfComposite*  m_scene;
    CIvfLight*      m_light;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    int m_beginX;
    int m_beginY;

public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
};
```

```
// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // Initialize variables

    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    int i, nSides;
    double r, angle, x, y;

    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 5.0, 5.0);

    // Create a materials

    CIvfMaterial* yellowMaterial = new CIvfMaterial();
    yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f);
    yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f);

    // Create scene composite

    m_scene = new CIvfComposite();

    // Create extrusion

    CIvfExtrusion* extrusion = new CIvfExtrusion();

    // Create section

    r = 0.5;
    nSides = 12;

    extrusion->setSectionSize(nSides + 1);

    for (i = 0; i<=nSides; i++)
    {
        angle = 2.0*M_PI*( ((double)i) / ((double)nSides) );
        x = r * cos(angle);
        y = r * sin(angle);
        extrusion->setSectionCoord(i, x, y);
        extrusion->setSectionNormal(i, x/r, y/r);
    }

    // Set spine

    extrusion->setSpineSize(6);
    extrusion->setSpineCoord(0,  0.5,  0.0,  1.5);
    extrusion->setSpineCoord(1,  1.0,  0.0,  1.0);
    extrusion->setSpineCoord(2,  1.0,  0.0, -1.0);
    extrusion->setSpineCoord(3, -1.0,  0.0, -1.0);
    extrusion->setSpineCoord(4, -1.0,  0.0,  1.0);
    extrusion->setSpineCoord(5, -0.5,  0.0,  1.5);

    // Set up-vector

    extrusion->setUpVector(0.0, 1.0, 0.0);

    // Set join style

    extrusion->setJoinStyle(TUBE_NORM_EDGE|TUBE_JN_ANGLE|TUBE_JN_CAP);

    // Set other properties

    extrusion->setMaterial(yellowMaterial);
    m_scene->addChild(extrusion);

    // First point

    CIvfAxis* axis = new CIvfAxis();
    m_scene->addChild(axis);

    // Create a light

    m_light = new CIvfLight();
    m_light->setPosition(1.0, 1.0, 1.0, 0.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setPerspective(45.0, 0.1, 100.0);
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;
        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);
        redraw();
    }

    if (isRightButtonDown())
    {
        if (getModifierKey() == CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }
        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);

        redraw();
    }
}

// ------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Extrusion example");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;
    return 0;
}
```

B.8 Level of detail and switch objects

```
// ------------------------------------------------------------
//
// Ivf++ LOD/Switch example
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfLOD.h>
#include <ivf/IvfCube.h>
#include <ivf/IvfCylinder.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:

    // Camera movement state variables

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    int m_beginX;
    int m_beginY;
    // Ivf++ object declarations

    CIvfCamera*    m_camera;
    CIvfLight*     m_light;
    CIvfComposite* m_scene;

    CIvfLOD*        m_lod1;
    CIvfLOD*        m_lod2;
    CIvfLOD*        m_lod3;
    CIvfLOD*        m_lod4;
    CIvfSwitch*     m_switch;

    double m_lodNear;
    double m_lodFar;

    void updateLOD();
public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
    virtual void onKeyboard(int key, int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // Initialize variables

    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    // Initialise LOD limits

    m_lodNear = 2.0;
    m_lodFar = 20.0;
    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 5.0, 20.0);

    // Create a materials

    CIvfMaterial* material = new CIvfMaterial();
    material->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    material->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);

    // Create scene

    m_scene = new CIvfComposite();

    // Create four spheres with different complexity

    CIvfSphere* sphereDetailed = new CIvfSphere();
    sphereDetailed->setSlices(12);
    sphereDetailed->setStacks(12);

    CIvfSphere* sphereNormal = new CIvfSphere();
    sphereNormal->setSlices(10);
    sphereNormal->setStacks(8);

    CIvfSphere* sphereSmall = new CIvfSphere();
    sphereSmall->setSlices(8);
    sphereSmall->setStacks(6);

    CIvfSphere* sphereTiny = new CIvfSphere();
    sphereTiny->setSlices(6);
    sphereTiny->setStacks(4);

    // Create LOD objects

    m_lod1 = new CIvfLOD();
    m_lod1->addChild(sphereDetailed);
    m_lod1->addChild(sphereNormal);
    m_lod1->addChild(sphereSmall);
    m_lod1->addChild(sphereTiny);
    m_lod1->setPosition(-5.0, 0.0, -5.0);
    m_lod1->setMaterial(material);
    m_lod1->setCamera(m_camera);
    m_lod1->setLimits(m_lodNear, m_lodFar);
    m_scene->addChild(m_lod1);

    m_lod2 = new CIvfLOD();
    m_lod2->addChild(sphereDetailed);
    m_lod2->addChild(sphereNormal);
    m_lod2->addChild(sphereSmall);
    m_lod2->addChild(sphereTiny);
    m_lod2->setPosition(5.0, 0.0, -5.0);
    m_lod2->setMaterial(material);
    m_lod2->setCamera(m_camera);
    m_lod2->setLimits(m_lodNear, m_lodFar);
    m_scene->addChild(m_lod2);

    m_lod3 = new CIvfLOD();
    m_lod3->addChild(sphereDetailed);
    m_lod3->addChild(sphereNormal);
    m_lod3->addChild(sphereSmall);
    m_lod3->addChild(sphereTiny);
    m_lod3->setPosition(-5.0, 0.0, 5.0);
    m_lod3->setMaterial(material);
    m_lod3->setCamera(m_camera);
    m_lod3->setLimits(m_lodNear, m_lodFar);
    m_scene->addChild(m_lod3);

    m_lod4 = new CIvfLOD();
    m_lod4->addChild(sphereDetailed);
    m_lod4->addChild(sphereNormal);
    m_lod4->addChild(sphereSmall);
    m_lod4->addChild(sphereTiny);
    m_lod4->setPosition(5.0, 0.0, 5.0);
    m_lod4->setCamera(m_camera);
    m_lod4->setMaterial(material);
    m_lod4->setLimits(m_lodNear, m_lodFar);
    m_scene->addChild(m_lod4);

    // Create a switch object

    CIvfCube* cube = new CIvfCube();
    cube->setMaterial(material);
    CIvfCylinder* cylinder = new CIvfCylinder();
    cylinder->setMaterial(material);

    m_switch = new CIvfSwitch();
    m_switch->addChild(cube);
    m_switch->addChild(cylinder);

    m_scene->addChild(m_switch);

    // Create a light

    m_light = new CIvfLight();
    m_light->setPosition(1.0, 1.0, 1.0, 0.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
```

```
{
    m_camera->setPerspective(45.0, 0.1, 100.0);
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);

        redraw();
    }
    if (isRightButtonDown())
    {
        if (getModifierKey() == CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }
        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);

        redraw();
    }
}

// ------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    switch (key) {
    case 'a':
        m_lodNear += 0.5;
        updateLOD();
        redraw();
        break;
    case 'z':
        m_lodNear -= 0.5;
        updateLOD();
        redraw();
        break;
    case 's':
        m_lodFar += 0.5;
        updateLOD();
        redraw();
        break;
    case 'x':
        m_lodFar -= 0.5;
        updateLOD();
        redraw();
        break;
    case 'd':
        m_switch->cycleForward();
        redraw();
        break;
    default:
        break;
    }
}

// ------------------------------------------------------------
void CExampleWindow::updateLOD()
{
    m_lod1->setLimits(m_lodNear, m_lodFar);
    m_lod2->setLimits(m_lodNear, m_lodFar);
    m_lod3->setLimits(m_lodNear, m_lodFar);
    m_lod4->setLimits(m_lodNear, m_lodFar);

    cout << "Lod limits set to: " << m_lodNear << ", " << m_lodFar << endl;
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ LOD/Switch example");
    window->show();

    // Enter main application loop

    app->run();
    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.9 Scene graph culling

```
// ------------------------------------------------------------
//
// Ivf++ Scene graph culling example
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfTransform.h>
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCulling.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:

    // Camera movement state variables

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    int m_beginX;
    int m_beginY;

    // Ivf++ object declarations

    CIvfCamera*          m_camera;
    CIvfCamera*          m_externalCamera;
    CIvfComposite*       m_scene;
    CIvfCulling*         m_culling;
    CIvfLight*           m_light;
    CIvfCamera*          m_currentCamera;

    bool m_useCulling;

public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
    virtual void onKeyboard(int key, int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // Initialize variables

    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    // Initialise culling setting

    m_useCulling = true;

    // Create cameras

    m_camera = new CIvfCamera();
    m_camera->addReference();
    m_camera->setPosition(0.0, 0.0, 10.0);
```

```
m_externalCamera = new CIvfCamera();
m_externalCamera->addReference();
m_externalCamera->setPosition(25.0, 50.0, 50.0);

m_currentCamera = m_camera;

// Create a materials

CIvfMaterial* material = new CIvfMaterial();
material->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
material->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);

// Create scene

m_scene = new CIvfComposite();

// Create a somewhat complex sphere

int i, j, k;
int nNodes = 8;
double distance = 20.0/(nNodes-1);

// Geometry is reduced by reusing the same sphere.

CIvfSphere* sphere = new CIvfSphere();
sphere->setSlices(12);
sphere->setStacks(12);
sphere->setMaterial(material);
sphere->setRadius(0.5);

// Create a grid of spheres.

CIvfTransform* arrayXlt = new CIvfTransform();
CIvfTransform* xlt;

for (i=0; i<nNodes; i++)
    for (j=0; j<nNodes; j++)
        for (k=0; k<nNodes; k++)
        {
            xlt = new CIvfTransform();
            xlt->setPosition(
                -10 + distance*i,
                -10 + distance*j,
                -10 + distance*k);
            xlt->addChild(sphere);
            arrayXlt->addChild(xlt);
        }

arrayXlt->setPosition(0.0, 0.0, -15.0);
```

```
    arrayXlt->setRotationQuat(0.0, 0.0, 1.0, 45.0);

    m_scene->addChild(arrayXlt);

    // Create culling object

    m_culling = new CIvfCulling();
    m_culling->setComposite(m_scene);
    m_culling->setCullView(m_camera);

    // Create a light

    m_light = new CIvfLight();
    m_light->setType(CIvfLight::LT_DIRECTIONAL);
    m_light->setDirection(1.0, 1.0, 1.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    if (m_useCulling)
        m_culling->cull();

    m_light->render();
    m_currentCamera->render();
    m_scene->render();

    cout << "Culled objects = " << m_culling->getCullCount() << endl;;
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_light;
    delete m_culling;
    delete m_scene;
    delete m_camera;
    delete m_externalCamera;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);

        redraw();
    }

    if (isRightButtonDown())
    {
        if (getModifierKey() == CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }
        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);

        redraw();
    }
}

// ------------------------------------------------------------
```

```
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
void CExampleWindow::onKeyboard(int key, int x, int y)
{
    switch (key) {
    case 'c':
        if (m_useCulling)
        {
            cout << "culling off" << endl;
            m_useCulling = false;
        }
        else
        {
            cout << "culling on" << endl;
            m_useCulling = true;
        }

        redraw();
        break;
    case 'x':
        if (m_currentCamera==m_camera)
        {
            m_currentCamera = m_externalCamera;
        }
        else
        {
            m_currentCamera = m_camera;
        }
        redraw();
        break;
    default:
        break;
    }
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.
    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Scene-graph culling example");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.10 Object selection

```
// ------------------------------------------------------------
//
// Ivf++ Object selection example
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfCube.h>
#include <ivf/IvfSphere.h>
#include <ivf/IvfCylinder.h>
#include <ivf/IvfCone.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfBufferSelection.h>
```

```
#include <ivf/IvfLight.h>
#include <ivf/IvfMaterial.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:

    // Ivf++ objects

    CIvfCamera*             m_camera;
    CIvfComposite*          m_scene;
    CIvfLight*              m_light;

    CIvfBufferSelection*    m_selection;
public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onMouseDown(int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(0.0, 4.0, 9.0);

    // Create a materials

    CIvfMaterial* redMaterial = new CIvfMaterial();
    redMaterial->setDiffuseColor(1.0f, 0.0f, 0.0f, 1.0f);
    redMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    redMaterial->setAmbientColor(0.5f, 0.0f, 0.0f, 1.0f);

    CIvfMaterial* greenMaterial = new CIvfMaterial();
    greenMaterial->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    greenMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    greenMaterial->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);
    CIvfMaterial* blueMaterial = new CIvfMaterial();
    blueMaterial->setDiffuseColor(0.0f, 0.0f, 1.0f, 1.0f);
    blueMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    blueMaterial->setAmbientColor(0.0f, 0.0f, 0.5f, 1.0f);

    CIvfMaterial* yellowMaterial = new CIvfMaterial();
    yellowMaterial->setDiffuseColor(1.0f, 1.0f, 0.0f, 1.0f);
    yellowMaterial->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    yellowMaterial->setAmbientColor(0.5f, 0.5f, 0.0f, 1.0f);

    // Create scene composite

    m_scene = new CIvfComposite();

    // Create objects

    CIvfCube* cube = new CIvfCube();
    cube->setMaterial(redMaterial);
    cube->setPosition(2.0, 0.0, 2.0);
    m_scene->addChild(cube);

    CIvfSphere* sphere = new CIvfSphere();
    sphere->setMaterial(greenMaterial);
    sphere->setPosition(-2.0, 0.0, 2.0);
    m_scene->addChild(sphere);

    CIvfCylinder* cylinder = new CIvfCylinder();
    cylinder->setMaterial(blueMaterial);
    cylinder->setPosition(-2.0, 0.0, -2.0);
    m_scene->addChild(cylinder);

    CIvfCone* cone = new CIvfCone();
    cone->setMaterial(yellowMaterial);
    cone->setPosition(2.0, 0.0, -2.0);
    cone->setRotationQuat(0.0, 0.0, 1.0, 45.0);
    m_scene->addChild(cone);

    CIvfAxis* axis = new CIvfAxis();
    m_scene->addChild(axis);

    // Setup the selection algorithm,

    m_selection = new CIvfBufferSelection();
    m_selection->setView(m_camera);
    m_selection->setComposite(m_scene);

    // Create a light

    m_light = new CIvfLight();
    m_light->setType(CIvfLight::LT_DIRECTIONAL);
    m_light->setDirection(1.0, 1.0, 1.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);
}

// ------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setPerspective(45.0, 0.1, 100.0);
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// ------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_camera->render();
    m_light->render();
    m_scene->render();
}

// ------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
    delete m_selection;
}

// ------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    if (isLeftButtonDown())
    {
        m_selection->pick(x, y);
        if (m_selection->getSelectedShape()!=NULL)
        {
            m_selection->getSelectedShape()->setHighlight(IVF_HIGHLIGHT_ON);
            redraw();
        }
        else
        {
            m_scene->setHighlightChildren(IVF_HIGHLIGHT_OFF);
            redraw();
        }
    }
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------
```

```
int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Selection example");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```

B.11 Loading 3d models

```
// ------------------------------------------------------------
//
// Ivf++ Ac3D file reader example
//
// ------------------------------------------------------------
//
// Author: Jonas Lindemann
//

// ------------------------------------------------------------
// Include files
// ------------------------------------------------------------

#include <ivfui/IvfApplication.h>
#include <ivfui/IvfWindow.h>

#include <ivf/IvfCamera.h>
#include <ivf/IvfAxis.h>
#include <ivf/IvfComposite.h>
#include <ivf/IvfLight.h>
```

```
#include <ivffile/IvfAc3DReader.h>

// ------------------------------------------------------------
// Window class definition
// ------------------------------------------------------------

class CExampleWindow: public CIvfWindow {
private:
    CIvfCamera*     m_camera;
    CIvfComposite*  m_scene;
    CIvfLight*      m_light;

    double m_angleX;
    double m_angleY;
    double m_moveX;
    double m_moveY;
    double m_zoomX;
    double m_zoomY;

    int m_beginX;
    int m_beginY;

    bool m_rotating;

public:
    CExampleWindow(int X, int Y, int W, int H)
        :CIvfWindow(X, Y, W, H) {};

    virtual void onInit(int width, int height);
    virtual void onResize(int width, int height);
    virtual void onRender();
    virtual void onDestroy();
    virtual void onMouseDown(int x, int y);
    virtual void onMouseMove(int x, int y);
    virtual void onMouseUp(int x, int y);
    virtual bool onTimeout0();
    virtual void onKeyboard(int key, int x, int y);
};

// ------------------------------------------------------------
// Window class implementation
// ------------------------------------------------------------

void CExampleWindow::onInit(int width, int height)
{
    // Initialize variables

    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    m_rotating = false;

    // Initialize Ivf++ camera

    m_camera = new CIvfCamera();
    m_camera->setPosition(-0.0, 3.0, 5.0);
    m_camera->setTarget(-0.0, 0.0, 0.0);

    // Create a materials

    CIvfMaterial* material = new CIvfMaterial();
    material->setDiffuseColor(0.0f, 1.0f, 0.0f, 1.0f);
    material->setSpecularColor(1.0f, 1.0f, 1.0f, 1.0f);
    material->setAmbientColor(0.0f, 0.5f, 0.0f, 1.0f);

    // Create scene

    m_scene = new CIvfComposite();

    // Create a file reader

    CIvfAc3DReader* acReader = new CIvfAc3DReader();

    // Set parameters

    acReader->setFileName("models/groups.ac");
    acReader->setScaling(1.0);

    // Read file

    acReader->read();

    // Retrieve poly set

    CIvfShape* shape = acReader->getShape();

    m_scene->addChild(shape);

    delete acReader;

    // Create a light

    m_light = new CIvfLight();
    m_light->setPosition(1.0, 1.0, 1.0, 0.0);
    m_light->setAmbient(0.2f, 0.2f, 0.2f, 1.0f);

    enableTimeout(0.01, 0);
}
```

```
// --------------------------------------------------------------
void CExampleWindow::onResize(int width, int height)
{
    m_camera->setPerspective(45.0, 0.1, 100.0);
    m_camera->setViewPort(width, height);
    m_camera->initialize();
}

// --------------------------------------------------------------
void CExampleWindow::onRender()
{
    m_light->render();
    m_camera->render();
    m_scene->render();
}

// --------------------------------------------------------------
void CExampleWindow::onDestroy()
{
    delete m_camera;
    delete m_light;
    delete m_scene;
}

// --------------------------------------------------------------
void CExampleWindow::onMouseDown(int x, int y)
{
    m_beginX = x;
    m_beginY = y;
}

// --------------------------------------------------------------
void CExampleWindow::onMouseMove(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;

    if (isLeftButtonDown())
    {
        m_angleX = (x - m_beginX);
        m_angleY = (y - m_beginY);
        m_beginX = x;
        m_beginY = y;

        m_camera->rotatePositionY(m_angleX/100.0);
        m_camera->rotatePositionX(m_angleY/100.0);
        redraw();
    }

    if (isRightButtonDown())
    {
        if (getModifierKey() == CIvfWidgetBase::MT_SHIFT)
        {
            m_zoomX = (x - m_beginX);
            m_zoomY = (y - m_beginY);
        }
        else
        {
            m_moveX = (x - m_beginX);
            m_moveY = (y - m_beginY);
        }
        m_beginX = x;
        m_beginY = y;

        m_camera->moveSideways(m_moveX/100.0);
        m_camera->moveVertical(m_moveY/100.0);
        m_camera->moveDepth(m_zoomY/50.0);

        redraw();
    }
}

// ------------------------------------------------------------
void CExampleWindow::onMouseUp(int x, int y)
{
    m_angleX = 0.0;
    m_angleY = 0.0;
    m_moveX = 0.0;
    m_moveY = 0.0;
    m_zoomX = 0.0;
    m_zoomY = 0.0;
}

// ------------------------------------------------------------
bool CExampleWindow::onTimeout0()
{
    if (m_rotating)
    {
        m_angleX = 0.1;
        redraw();
    }
    return true;
}

// ------------------------------------------------------------
void CExampleWindow::onKeyboard(int key, int x, int y)
```

```
{
    switch (key) {
    case 'r' :
        if (m_rotating)
            m_rotating = false;
        else
            m_rotating = true;
        break;
    default:
        break;
    }
}

// ------------------------------------------------------------
// Main program
// ------------------------------------------------------------

int main(int argc, char **argv)
{
    // Create Ivf++ application object.

    CIvfApplication* app = new CIvfApplication(IVF_DOUBLE|IVF_RGB);

    // Create a window

    CExampleWindow* window = new CExampleWindow(0, 0, 512, 512);

    // Set window title and show window

    window->setWindowTitle("Ivf++ Ac3D file reader example");
    window->show();

    // Enter main application loop

    app->run();

    // Clean up and return

    delete app;
    delete window;

    return 0;
}
```