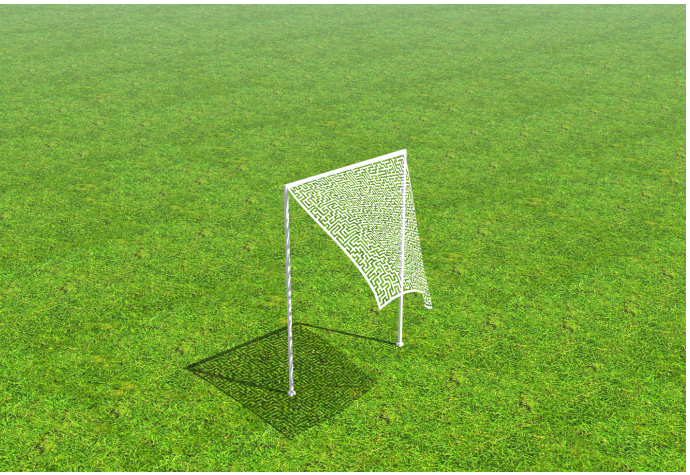
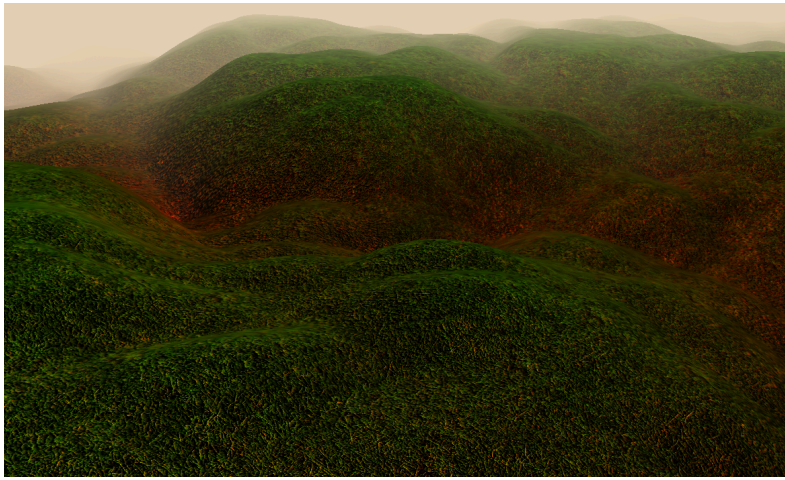
COSC342: Computer Graphics

2017





Tutorial

C++ Recap

Stefanie Zollmann

C++ CLASSES

```
class Camera{
public:
    //! Default constructor
    /*! Setting up default camera. */
    Camera();

    /*!  Set position. */
    void setPosition(glm::vec3 pos);
    //..

private:
    // camera pose parameters
    glm::vec3 m_position;           //!< Position of camera
    glm::vec3 m_lookat;             //!< Lookat vector of camera
}
```

C++ OBJECTS

```
Camera myCamera1; // Memory allocation on stack
myCamera1.setPosition(glm::vec3(0,0,-5));

Camera* myCamera2 = new Camera(); // Memory allocation on heap
myCamera2->setPosition(glm::vec3(0,0,-5));
```

C++ OBJECTS

```
Camera myCamera1; // Memory allocation on stack
myCamera1.setPosition(glm::vec3(0,0,-5));

Camera* myCamera2 = new Camera(); // Memory allocation on heap
myCamera2->setPosition(glm::vec3(0,0,-5));

// myCamera2 is on heap - we need to take of memory deallocation
// Every call to new must be matched by a call to delete; failure to do
so causes memory leaks.
delete myCamera2;
```

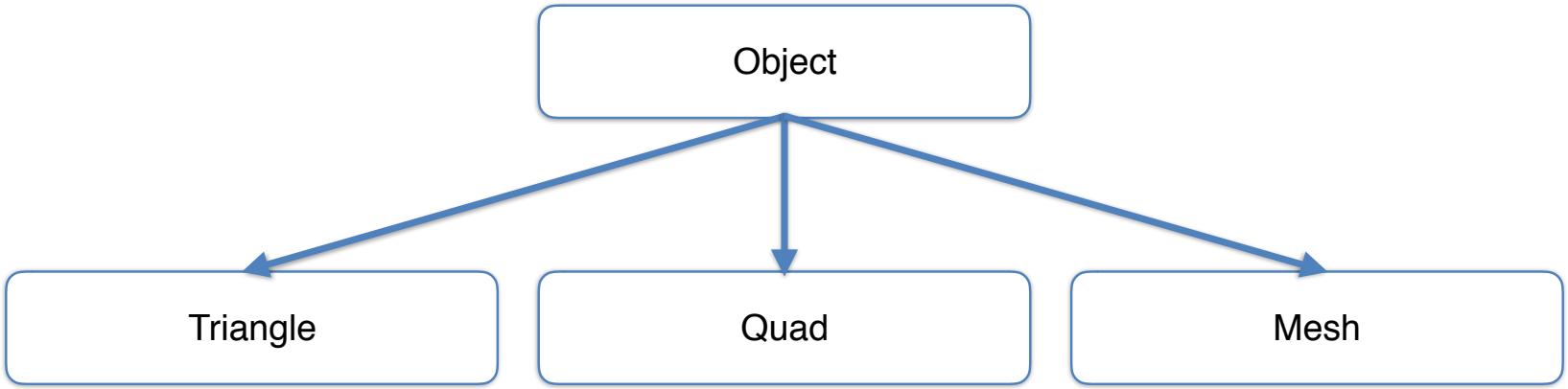
DATA ENCAPSULATION IN C++

```
class Camera{
public:
    //! Default constructor
    /*! Setting up default camera. */
    Camera();

    /*!  Set position. */
    void setPosition(glm::vec3 pos);
    //..

private:
    // camera pose parameters
    glm::vec3 m_position;            //!< Position of camera
    glm::vec3 m_lookat;              //!< Lookat vector of camera
}
```

ACCESSING THE DATA MEMBERS

```
Camera myCamera1;
myCamera1.setPosition(glm::vec3(0,0,-5));
myCamera1.position = glm::vec3(0,0,-5); // not allowed - private member


Camera* myCamera2 = new Camera();
myCamera2->setPosition(glm::vec3(0,0,-5));
myCamera2->position= glm::vec3(0,0,-5); // not allowed - private member
```

C++ INHERITANCE




```
class DerivedClass: access-specifier BaseClass
```

Quad Example:

```
class Quad: public Object
```

C++ INHERITANCE

```
class Object{
    public:
        Object();
        //! Destructor
        /*! Delete all related resources. */
        virtual ~Object();
        void setShader(Shader* newshader);
        glm::mat4 getTransform();
         void addTransform(glm::mat4 mat);
        virtual void render(Camera* camera)=0;
        void setTranslate(glm::vec3 translateVec);
        void setScale(float scale);
        void bindShaders();

    private:
        std::string name;       //!< name of object
        glm::mat4 transform;    //!< transform matrix of object
        glm::mat4 rotMat;       //!< rotation matrix
        glm::mat4 transMat;     //!< translation matrix
        glm::mat4 scaleMat;     //!< scale matrix

    protected:
        Shader* shader;         //!< object can have a shader
};
```

```
class Quad:  public Object{

    public:
        //! Default constructor
        /*! Setting up default quad. */
        Quad();
        //! Destructor
        /*! Delete quad. */
        ~Quad();
        //! init
        /*! Setting up default quad. */
        void init();
        //! render
        /*! Render default quad. */
        void render(Camera* camera);
    private:

        GLfloat g_vertex_buffer_data[18]; //vertices of Quad
        GLuint vertexbuffer;

};
```

C++ INHERITANCE

```
class Object{
    public:
        Object();
        //! Destructor
        /*! Delete all related resources. */
        virtual ~Object();
        void setShader(Shader* newshader);
        glm::mat4 getTransform();
         void addTransform(glm::mat4 mat);
        virtual void render(Camera* camera)=0;
        void setTranslate(glm::vec3 translateVec);
        void setScale(float scale);
        void bindShaders();

    private:
        std::string name;       //!< name of object
        glm::mat4 transform;    //!< transform matrix of object
        glm::mat4 rotMat;       //!< rotation matrix
        glm::mat4 transMat;     //!< translation matrix
        glm::mat4 scaleMat;     //!< scale matrix

    protected:
        Shader* shader;         //!< object can have a shader
};
```

```
class Triangle:  public Object{

    public:

        //! Default constructor
        /*! Setting up default triangle. */
        Triangle();
        //! Destructor
        /*! Delete triangle. */
        ~Triangle();
        //! init
        /*! Setting up default triangle. */
        void init();
        //! render
        /*! Render default quad. */
        void render(Camera* camera);

    private:

        GLfloat g_vertex_buffer_data[9];
        GLuint vertexbuffer;

};
```

C++ OVERLOADING

```
void Triangle::render(Camera* camera){
    bindShaders();
    // Build the model matrix -get from object
    glm::mat4 ModelMatrix = this->getTransform();
    glm::mat4 MVP = camera->getViewProjectionMatrix() * ModelMatrix;
    // Send our transformation to the currently bound shader,
    // in the "MVP" uniform
    shader->updateMVP(MVP);
    // 1rst attribute buffer : vertices
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glVertexAttribPointer(
                          0,                  // attribute 0. No particular reason for 0, but must
match the layout in the shader.
                          3,                  // size
                          GL_FLOAT,           // type
                          GL_FALSE,           // normalized?
                          0,                  // stride
                          (void*)0            // array buffer offset
                          );

    // Draw the triangle !
    glDrawArrays(GL_TRIANGLES, 0, 3); // 3 indices starting at 0 -> 1 triangle

    glDisableVertexAttribArray(0);
}
```

C++ OVERLOADING

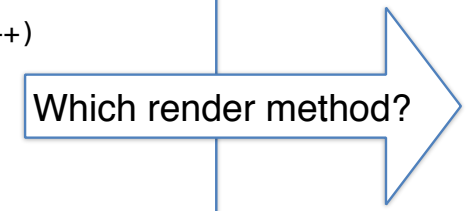
```
void Quad::render(Camera* camera){
    bindShaders();
    // Build the model matrix -get from object
    glm::mat4 ModelMatrix = this->getTransform();
    glm::mat4 MVP = camera->getViewProjectionMatrix() * ModelMatrix;
    // Send our transformation to the currently bound shader,
    // in the "MVP" uniform
    shader->updateMVP(MVP);
    // 1rst attribute buffer : vertices
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glVertexAttribPointer(
                          0,                  // attribute 0. No particular reason for 0, but must
match the layout in the shader.
                          3,                  // size
                          GL_FLOAT,           // type
                          GL_FALSE,           // normalized?
                          0,                  // stride
                          (void*)0            // array buffer offset
                          );

    // Draw the quad with two triangles !
    glDrawArrays(GL_TRIANGLES, 0, 6); // 6 indices starting at 0 -> creates 2 triangles -> 1 quad
    glDisableVertexAttribArray(0);
}
```

POLYMORPHISM IN C++

```
Scene* myScene = new Scene();
Triangle* myTriangle = new Triangle();
myScene->addObject(myTriangle);
Quad* myQuad = new Quad();
myScene->addObject(myQuad);
myScene->render(myCamera);
```

Main.cpp

```
void Scene::render(Camera* camera){
    for (int i=0;i<sceneObjects.size();i++)
    {
        sceneObjects[i]->render(camera);
    }
}

void Scene::addObject(Object *object){
    sceneObjects.push_back(object);

}
```



Scene.cpp

POLYMORPHISM IN C++

```
Scene* myScene = new Scene();
Triangle* myTriangle = new Triangle();
myScene->addObject(myTriangle);
Quad* myQuad = new Quad();
myScene->addObject(myQuad);
myScene->render(myCamera);
```

Main.cpp

```
void Scene::render(Camera* camera){
    for (int i=0;i<sceneObjects.size();i++)
    {
        sceneObjects[i]->render(camera);
    }
}

void Scene::addObject(Object *object){
    sceneObjects.push_back(object);

}
```

Scene.cpp

Which render method?

Using late binding by using **virtual** for the render method in the base class (Object):

```
virtual void render(Camera* camera)=0;
```

POLYMORPHISM IN C++

```
Scene* myScene = new Scene();
Triangle* myTriangle = new Triangle();
myScene->addObject(myTriangle);
Quad* myQuad = new Quad();
myScene->addObject(myQuad);
myScene->render(myCamera);
```

Main.cpp

```
void Scene::render(Camera* camera){
    for (int i=0;i<sceneObjects.size();i++)
    {
        sceneObjects[i]->render(camera);
    }
}

void Scene::addObject(Object *object){
    sceneObjects.push_back(object);

}
```

Scene.cpp

Which render method?

Using late binding by using **virtual** for the render method in the base class (Object):

```
virtual void render(Camera* camera)=0;
```

If sceneObjects[i] is Quad: Quad's render method will be called

If sceneObjects[i] is Triangle: Triangle's render method will be called

ABSTRACT CLASS

```
class Object{
    public:
        Object();
        //! Destructor
        /*! Delete all related resources. */
        virtual ~Object();
        void setShader(Shader* newshader);
        glm::mat4 getTransform();
        void addTransform(glm::mat4 mat);
        virtual void render(Camera* camera)=0;
        void setTranslate(glm::vec3 translateVec);

    Object* myobject = new Object();

        std::string name;       //!< name of object
        glm::mat4 transform;    //!< transform matrix of object
        glm::mat4 rotMat;       //!< rotation matrix
        glm::mat4 transMat;     //!< translation matrix
        glm::mat4 scaleMat;     //!< scale matrix

    protected:
        Shader* shader;         //!< object can have a shader
};
```

Allocating an object of abstract class type 'Object'

Be careful, you can't create objects of abstract class type. Will create a compile error.

Thank You!

For more material visit

http://www.cs.otago.ac.nz/cosc342/