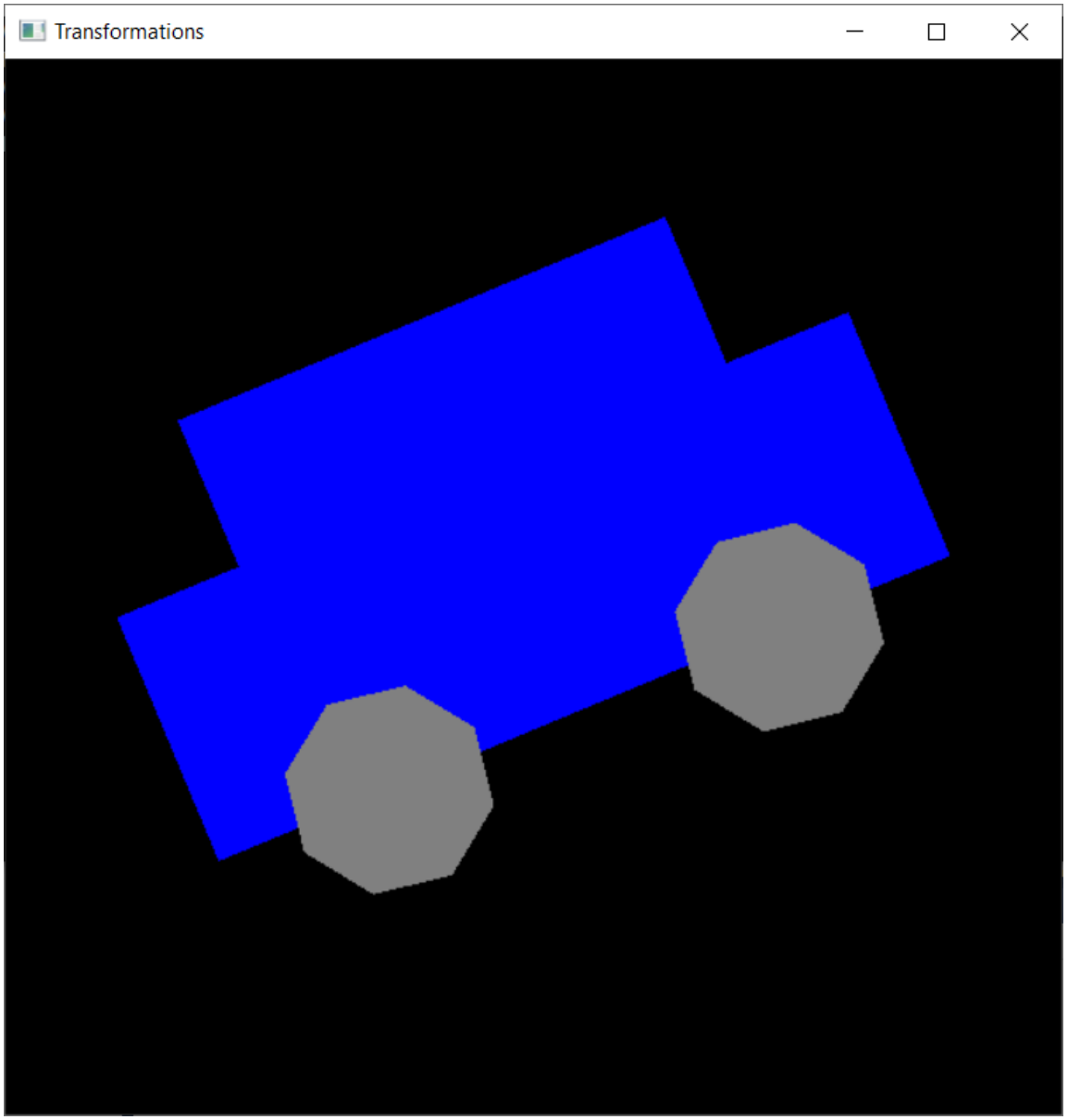
Computer Graphics Laboratory Exercise 3

Balázs Csébfalvi

http://cg.iit.bme.hu/portal/en/cgbme

Transforming hierarchical models



Use the original shader programs

GLSL shader programs

Vertex shader

```
#version 330
precision highp float;
uniform mat4 MVP;
layout(location = 0) in vec2 vp;
void main() {
  gl_Position = vec4(vp.x, vp.y, 0, 1) * MVP;
}
```

Pixel shader

```
#version 330
precision highp float;
uniform vec3 color;
out vec4 outColor;
void main() {
  outColor = vec4(color, 1);
}
```

Define square and circle shapes

```
class Shape { // abstract base class
protected:
        unsigned int vao;
public:
        virtual void Render() = 0;
};

class Square : public Shape {
public:
        Square() {
        //define VBO here
        }
        void Render() {
        // activate VAO, upload uniform color, render the square as a triangle strip
        }
};

class Circle  : public Shape {
static const int nvertices = 10;
public:
        Circle() {
        //define VBO here
        }
        void Render() {
        // activate VAO, upload uniform color, render the circle as a triangle fan
        }
};
```

Define a hierarchical model by a tree structure

- Each node is an instance of class Object
- Each node contains
  - a pointer to a Shape
  - a transformation matrix
  - a vector of object components (object pointers)
- Functions of class Object
  - Constructor, Destructor (deletes the object components)
  - void AddComponent(Object *component) - adds a new object component
  - void Render() – renders the corresponding shape and the components as well, **the transformation matrix is applied on the shape and on the components as well**

Use animated components

```
enum Animation { NOT_MOVING, ROTATING, UP_AND_DOWN };
class Object {
protected:
        Shape *shape;
        mat4 transformation;
        std::vector<Object*> components;
        Animation animation;
public:
  // . . .
  mat4 Transformation() { float alpha = 0.0;
    switch(animation) {
      case NOT_MOVING: return transformation;
      case ROTATING: alpha = M_PI * time / 1000.0;
        return RotationMatrix(alpha, vec3(0.0, 0.0, 1.0)) *
        transformation;
      case UP_AND_DOWN: alpha = M_PI / 4.0 * cos(time / 1000.0);
        return RotationMatrix(alpha, vec3(0.0, 0.0, 1.0)) *
        transformation;
    }
  }
};
```

Not an animated object

Rotating object (wheels)

Periodically rotating object (car going up and down)

Define class Car

```
class Car {
  Object* object;
  std::vector<Shape*> shapes;
public:
  Car() {
    Shape *circle = new Circle, *square = new Square;
    shapes.push_back(square); shapes.push_back(circle);
    object = new Object(ScaleMatrix(vec3(1.0, 1.0, 1.0)), 0, UP_AND_DOWN);
    object->AddComponent(new Object(ScaleMatrix(vec3(1.5, 0.5, 1.0)), square));
    object->AddComponent(new Object(ScaleMatrix(vec3(1.0, 0.3, 1.0)) *
      TranslateMatrix(vec3(0.0, 0.4, 0.0)), square));
    object->AddComponent(new Object(ScaleMatrix(vec3(0.4, 0.4, 1.0)) *
      TranslateMatrix(vec3(-0.4, -0.25, 0.0)), circle, ROTATING));
    object->AddComponent(new Object(ScaleMatrix(vec3(0.4, 0.4, 1.0)) *
      TranslateMatrix(vec3(0.4, -0.25, 0.0)), circle, ROTATING));
  }

  ~Car() {
    for(int i = 0; i < shapes.size(); i++) delete shapes[i];
    delete object;
  }

  void Render() {
    object->Render();
  }
};
```