HOCHSCHULE HANNOVER
UNIVERSITY OF APPLIED SCIENCES AND ARTS
–
Fakultät IV Wirtschaft und Informatik

SCG3: An Extensible Scene Graph Library for Teaching Computer Graphics along the Programmable Pipeline

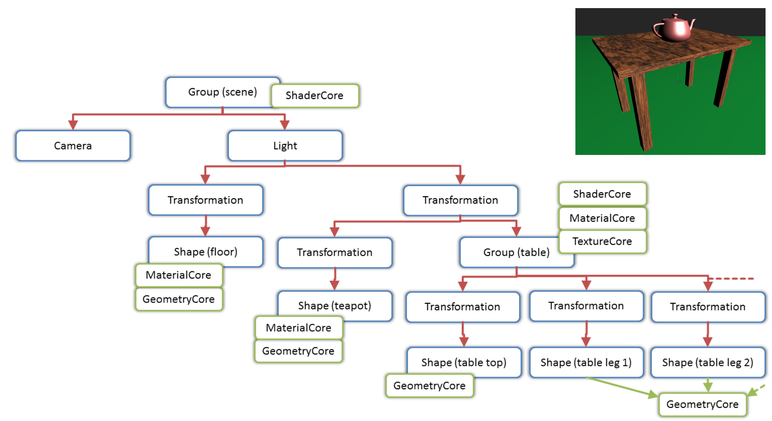
Volker Ahlers
University of Applied Sciences and Arts Hannover, Germany | Department of Computer Science

Aim: Teach Modern Computer Graphics!

- Computer graphics is typically enjoyed by students.
- Programming methods have significantly changed in recent years:
  - Integration of programmable shaders into graphics rendering pipeline (OpenGL, Direct3D)
  - Deprecation of fixed-function pipeline (but still widely used in computer graphics classes)
- What is the problem with teaching modern computer graphics?
  - Shaders required from first lesson on
  - Technical “overhead” for compiling shaders, defining buffer objects, computing transformation matrices, etc.

Enter Scene Graph Library

- Directed acyclic graph abstraction of 3D scene
  - Nodes forming tree structure with transformations, shapes, groups, lights, camera
  - Cores associated to one or more nodes, encapsulating geometry, shaders, materials, textures
  - Multiple render passes as graph traversal with different visitors
- Design patterns: visitor, composite, observer, strategy, decorator, …
- OpenGL 3.2 core profile with modern utility libraries (see references)
- ANSI C++11, including smart pointers and auto-typed variables
- Design objective: extensibility for student laboratory projects, implementing techniques and graphics effects learned in lecture



Put the Concept into Practice

- Weeks 1 to 3: OpenGL 3, shaders, and linear algebra review
- Weeks 4 to 14: scene graph and advanced graphics techniques, laboratory project in teams of 2 to 3 students
- Week 15: final project presentation and team examinations

```
// Scene graph application excerpt (C++11)
scg::ShaderCoreFactory shaderFactory("./shaders");
auto shaderPhong = shaderFactory.createShaderFromSourceFiles(
    {
      ShaderFile("phong_vert.glsl", GL_VERTEX_SHADER),
      ShaderFile("phong_frag.glsl", GL_FRAGMENT_SHADER),
      ShaderFile("lighting_blinn.glsl", GL_FRAGMENT_SHADER),
      ShaderFile("texture_none.glsl", GL_FRAGMENT_SHADER)
    });

scg::GeometryCoreFactory geometryFactory;
auto teapotCore = geometryFactory.createTeapot(0.35f);
auto teapot = scg::Shape::create();
teapot->addCore(materialRed)
      ->addCore(teapotCore);
auto teapotTrans = scg::Transformation::create();
teapotTrans->translate(glm::vec3(0.f, 0.9f, 0.f))
           ->rotate(-90.f, glm::vec3(1.f, 0.f, 0.f));

scene = scg::Group::create();
scene->addCore(shaderPhong);
scene->addChild(camera)
     ->addChild(light);
light->addChild(tableTrans);
tableTrans->addChild(table)
          ->addChild(teapotTrans);
teapotTrans->addChild(teapot);
```

```
// Phong vertex shader (GLSL)
in vec4 vVertex;                // ...
uniform mat4 modelViewMatrix;   // ...
smooth out vec3 ecVertex;       // ...

void main() {
  ecVertex = (modelViewMatrix * vVertex).xyz;
  ecNormal = normalMatrix * vNormal;
  gl_Position = mvpMatrix * vVertex;
  texCoord0 = textureMatrix * vTexCoord0;
}
```

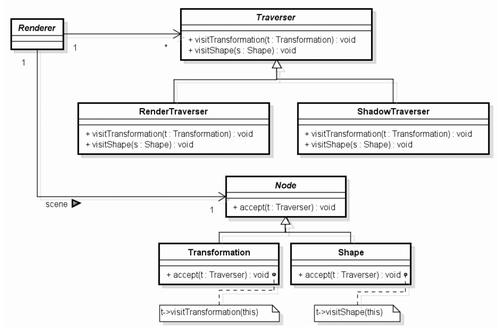
```
// Phong fragment shader (GLSL)
smooth in vec3 ecVertex;  // ...
out vec4 fragColor;

void applyLighting(...);  // defined in
vec4 applyTexture(...);   // other shaders

void main() {
  vec4 emissionAmbientDiffuse, specular;
  applyLighting(ecVertex, ecNormal,
       emissionAmbientDiffuse, specular);
  vec4 color = applyTexture(texCoord0,
       emissionAmbientDiffuse, specular);
  fragColor = clamp(color, 0., 1.);
}
```

Student Project PieRats

Kai Borchert, Uwe Brosch, Rudolf Podlich (1st year master)

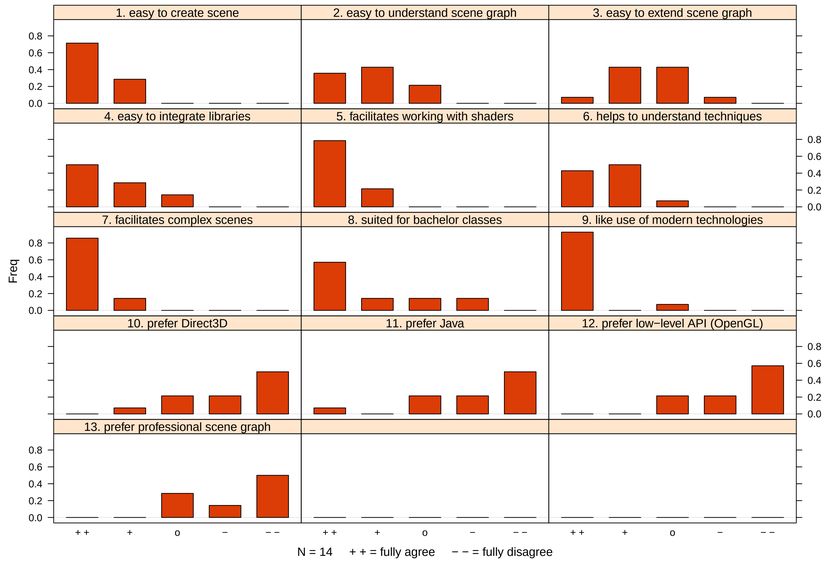
- Animation of ships with user interaction via game controllers
- Collision handling for ships
- Displacement mapping: tesselation shaders (OpenGL 4)
- Simulation of water waves: vertex and tesselation shaders
- Particle system: geometry shader

- Visitor pattern supports extensibility
- Example: `ShadowTraverser` can be added without modifying `Node` and derived classes.



What Do the Students Think?

- Evaluation in 1st and 2nd year master classes (scene graph library has not yet been used in bachelor classes)
- Main observations:
  - Scene graph facilitates working with shaders and understanding advanced graphics techniques
  - Using modern technologies is approved
  - Main criticism: missing tutorial for extending scene graph



References and Third-Party Libraries

E. Angel. Teaching computer graphics starting with shader-based OpenGL. In P. Cozzi, C. Riccio, editors, *OpenGL Insights*, pages 3–16. CRC Press, Boca Raton, FL, 2012.

E. Angel, D. Shreiner. Teaching a shader-based introduction to computer graphics. *IEEE Computer Graphics and Applications*, 31(2):9–13, 2011.

M. Bailey. Transitioning students to post-deprecation OpenGL. In P. Cozzi, C. Riccio, editors, *OpenGL Insights*, pages 17–26. CRC Press, Boca Raton, FL, 2012.

E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns.* Addison-Wesley, Boston, MA, 1995.

OpenGL specifications. `http://www.opengl.org/registry/`

GLFW: OpenGL window and event handling toolkit. `http://www.glfw.org/`

GLM: OpenGL mathematics library. `http://glm.g-truc.net/`

stb_image: C image loading library. `http://http://nothings.org/stb_image.c`

Acknowledgements: Ingo Ginkel, Frauke Sprengel, Henrik Tramberend, feedback of students

Contact: `volker.ahlers@hs-hannover.de`
Download: `https://github.com/vahlers/scg3`