JOHANNES KEPLER
UNIVERSITY LINZ
Institute of
Computer Graphics

Computer Graphics

Lab 3: Scene Graphs

Schedule

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

Lab 1	Introduction to WebGL	10.3.2017 / 14.3/2017	-
Lab 2	Transformations and Projections	21.3.2017 / 24.3.2017	Lecture: 14.3.2017
Lab 3	Scene Graphs	28.3.2017 / 31.3.2017	Lecture: 21.3.2017
Lab 4	Illumination and Shading	4.4.2017 / 7.4.2017	Lecture: 28.3.2017
Lab 5	Texturing	25.4.2017 / 28.4.2017	Lecture: 4.4.2017
Lab 6	Advanced Texture Mapping	2.5.2017 / 5.5.2017	Lecture: 26.4.2017
Lab 7a	CUDA	9.5.2017 / 12.5.2017	
Lab 7b	VTK	12.5.2017	

CG Lab Project: Create a Movie

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

30 seconds WebGL movie that shows scenes in a world

Use our framework

Implement:

- requirements & basic effects (e.g. camera, scenegraph, …)
- special effects of your choice

Detailed Specification:
www.cg.jku.at/teaching/computergraphics/lab

CG Lab Project: Create a Movie

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

Group project of 2 persons

Submissions:

08.05.2017: Announce group members and hand-in movie concept

-> you will get feedback!

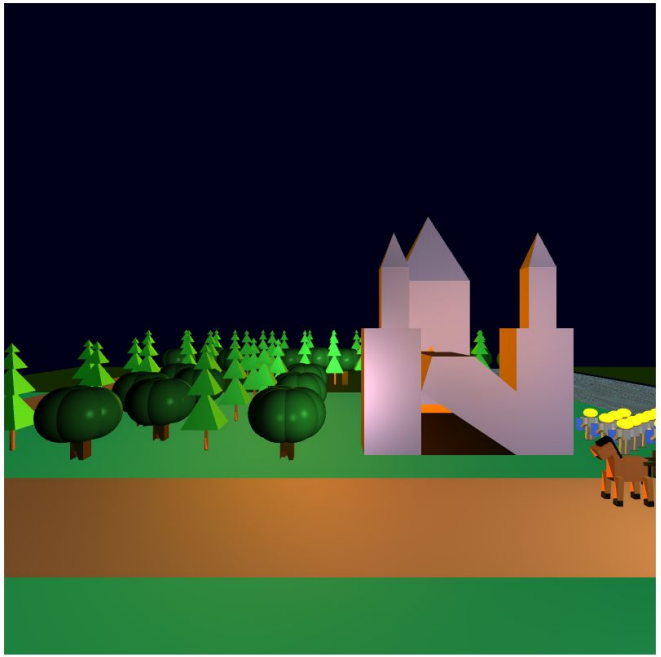
19.06.2017: Hand-in final package

Individual interviews (alone): **26.-30.06.2017**

CG Lab Project: Examples 2016

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics



Jose Illescas & Andi Avram-Anghel

Manfred Schlägl & Pieter-Jan Hoedt

Slides and lab material

www.cg.jku.at/teaching/computergraphics/lab

Recap

Transformation pipeline

Model-view transformations

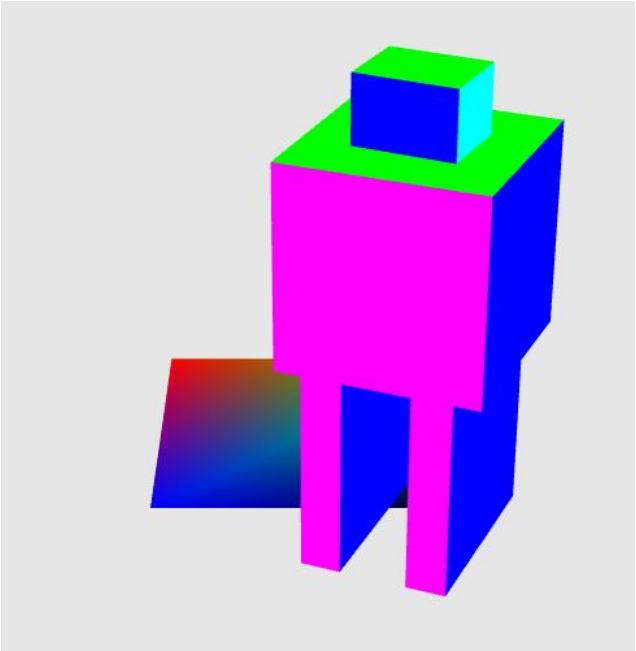
- Translate, scale, rotate, animations

Creating geometry using the index buffer

Projective transformations

- Orthographic and perspective projection

Camera transformations



Solution on GitHub:

- 02_transformations
- 02_transformations_handout

Agenda for Today

JOHANNES KEPLER UNIVERSITY LINZ

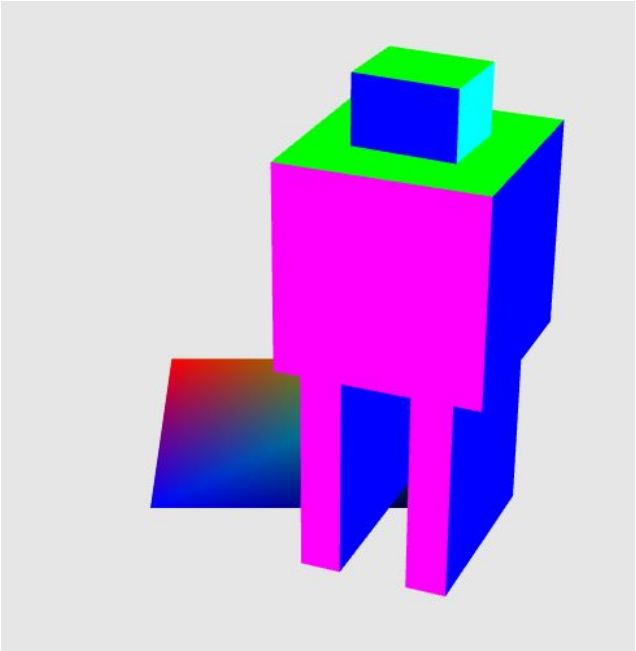
Institute of Computer Graphics

Depth Handling

Blending

Scene Graphs

- Abstraction into Nodes
- Scene graph traversal
- Base node class
- Render nodes
- Transformation nodes
- Implement robot using a scene graph



Fork me on GitHub

Dev Environment: Lab Package

JOHANNES KEPLER UNIVERSITY LINZ

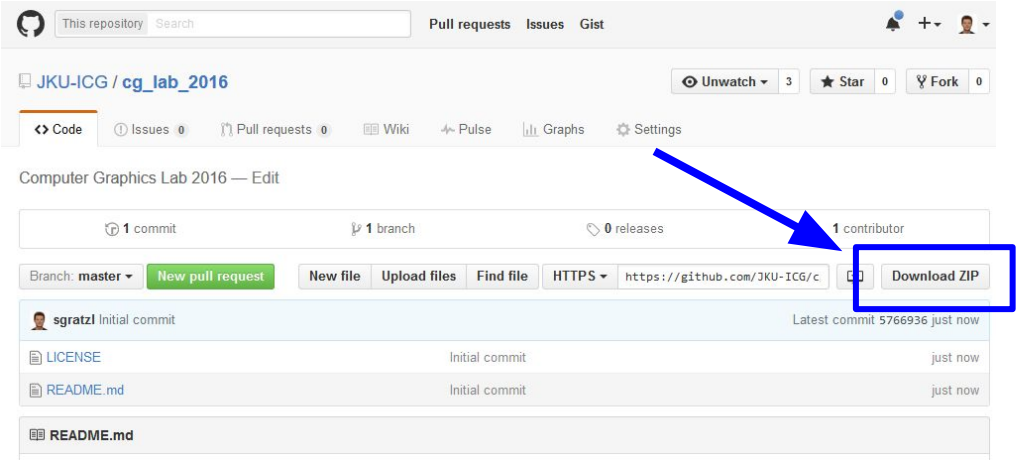
Institute of Computer Graphics

Hosted on GitHub: https://github.com/jku-icg/cg_lab_2017

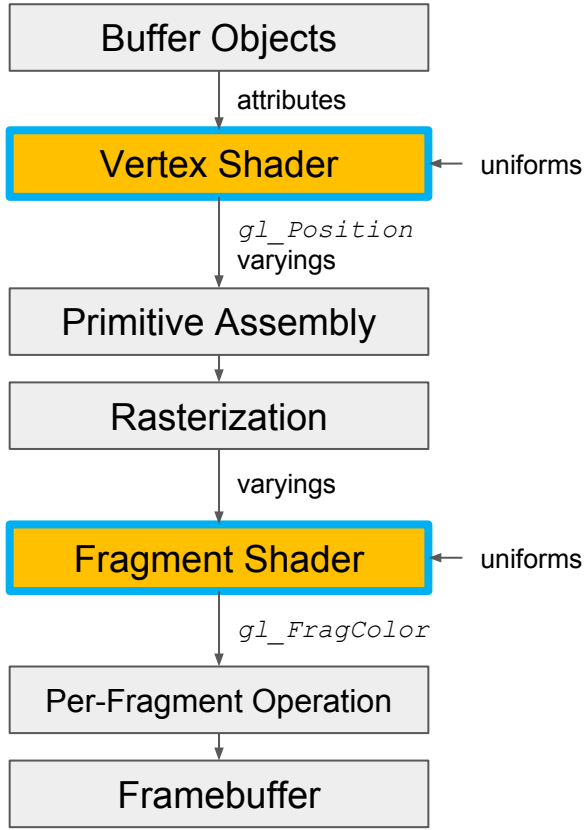
The repository will be updated during the lab with the new projects.

To get started (now):

1. Download the zip
2. Extract the folder
3. Open Atom editor
4. Use “Open Folder” in Atom
5. Start server on any port (Plugins -> Live Server)



Recap: Programmable Pipeline



Shader Parameter

- attributes
  streams for the vertex shader stored in buffer
  e.g., vertex position
- uniforms
  parameter from the program to the shader,
  e.g., static color, light position
- varyings
  out/input between shader stages with interpolation
  e.g., vertex color → fragment color

Recap: Structure of a WebGL Program

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

At initialization time

- Create all shaders and programs
- Create buffers and upload vertex data

At render time

- Set global states (enable depth testing, etc.)
- For each object you want to draw
  - Call `gl.useProgram` for the program needed to draw
  - Setup attributes for the object you want to draw
  - For each attribute call `gl.bindBuffer`, `gl.vertexAttribPointer`, `gl.enableVertexAttribArray`
  - Setup uniforms for the object you want to draw by calling `gl.uniformXXX`
  - Call `gl.drawArrays` or `gl.drawElements`

http://webglfundamentals.org/webgl/lessons/webgl-drawing-multiple-things.html

glMatrix Helper Library

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

Library for vector and matrix manipulation: http://glmatrix.net/

Library supports:

- Identity matrix, multiplication, inverse, clone, translation, rotation, scale, lookAt, orthographic & perspective projection, etc.

Already part of framework: libs/gl-matrix.js

Documentation: http://glmatrix.net/docs/2.2.0/

Use it from now on!

glMatrix Usage

Classes vec2, vec3, vec4, mat2, mat3, mat4, etc.

```
var identity = mat4.create();
var out = mat4.scale(mat4.create(),identity,[x, y, z]);
```

For convenience in our framework:

```
var out = glm.scale(x,y,z)
```

{mat4} mat4.scale(out, a, v)

Scales the mat4 by the dimensions in the given vec3

Parameters:

{mat4} **out**

the receiving matrix

{mat4} **a**

the matrix to scale

{vec3} **v**

the vec3 to scale the matrix by

Returns:

{mat4} out

Depth Handling

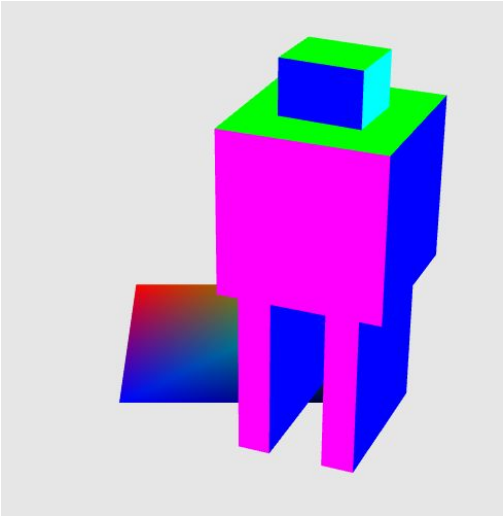
Allows OpenGL to decide which object is in front

Depth testing has to be enabled to tell OpenGL to perform a depth test and use the depth buffer

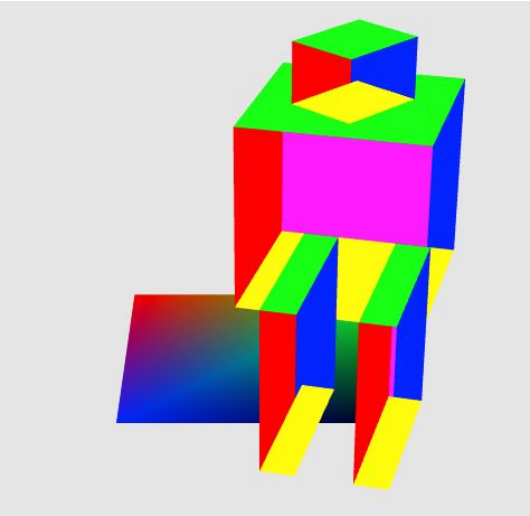
```
gl.enable(GL_DEPTH_TEST);
```

Try to turn it off!

But how does it work?



Depth testing enabled



Depth testing disabled

Depth Buffer (Z-Buffer)

JOHANNES KEPLER UNIVERSITY LINZ

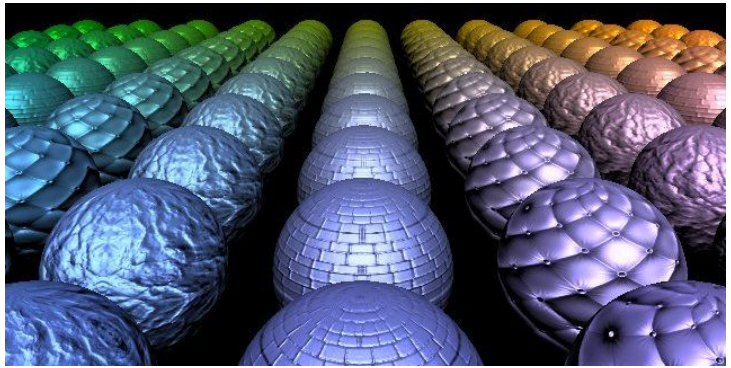
Institute of Computer Graphics

Contains normalized depth values (0-1) for all fragments

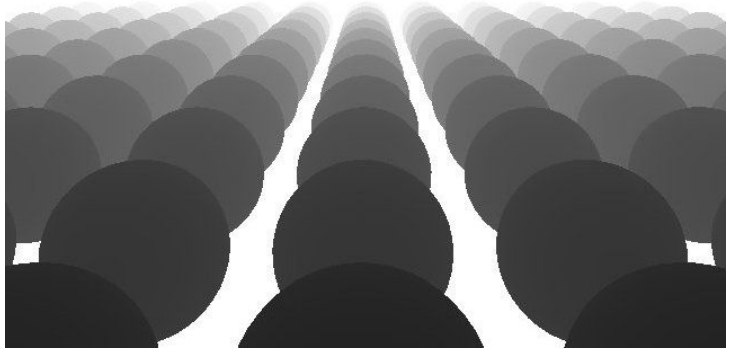
- OpenGL tests depth value of a fragment against the content of the depth buffer
- If test passes, the depth buffer is updated with the new depth value.
- If depth test fails, the fragment is discarded.

Depth test function

`gl.depthFunc(GL_LESS);` (the smaller value wins)



Sample scene



Corresponding depth buffer

http://romain.vergne.free.fr/teaching/IS/TP07.html

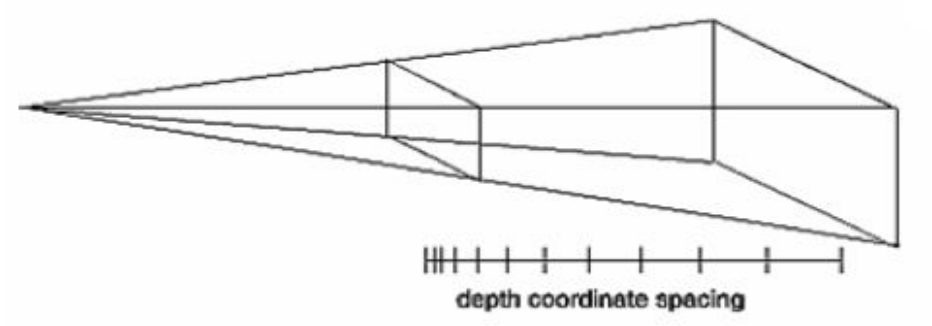
Z-Fighting

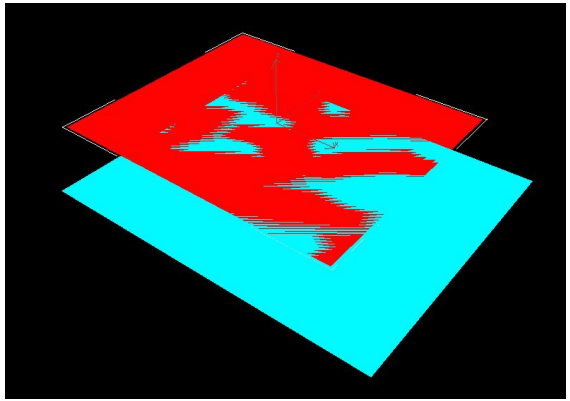
OpenGL can’t decide which face is in front

Happens when primitives are too close together

We can force it to happen

- By changing the near and far clipping planes’ values of the projection
- The closer to the near clipping plane, the denser the values get and therefore the further away, the coarser.





Blending: Alpha Values

JOHANNES KEPLER UNIVERSITY LINZ

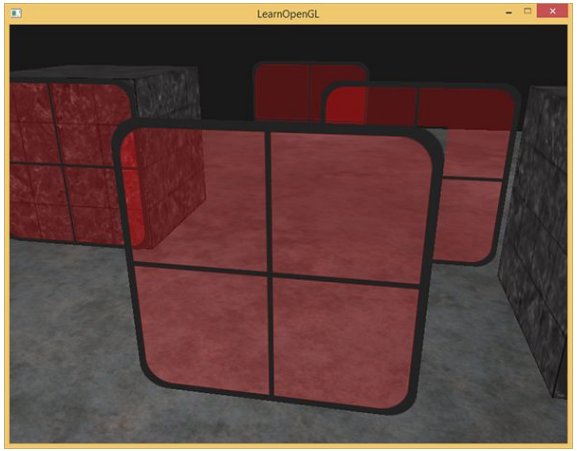
Institute of Computer Graphics

The alpha value defines an object's opacity

- 0 … transparent
- 1 … opaque
- (depending on blend function)

OpenGL offers various blend functions

- Defining how pixels are blended (see the RedBook for details)
- Each function defines a way of blending an incoming pixel (the source) with the currently stored one (the destination)
- `gl.blendFunc(source, destination)` sets the factors for source and destination
- Example: `gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);`



http://learnopengl.com/#!Advanced-OpenGL/Blending

Task 1: Blending

Goal: Make the robot semi-transparent (50%)

Step 1: Enable blending by calling `gl.enable(gl.BLEND);`

Step 2: Set blend function:

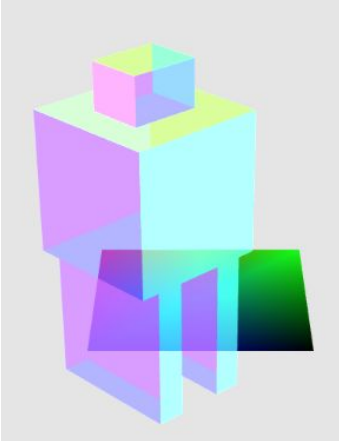
```
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
```

Step 3: Provide alpha value as uniform to fragment shader

- Use `gl.uniform1f` to set `u_alpha` to **0.5** for **robot**
- Use `gl.uniform1f` to set `u_alpha` to **1.0** for **quad**
- Usage: `gl.uniform1f(location, alpha)`

Step 4: Incorporate `u_alpha` in fragment shader

- Variable is already defined in fragment shader
- Transparency is fourth component of the color vector (RGBA)



Task 1: Solution

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

```
//TASK 1-1
gl.enable(gl.BLEND);
//TASK 1-2
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
```

```
//set alpha value for blending
//TASK 1-3
gl.uniform1f(gl.getUniformLocation(context.shader, "u_alpha"), 0.5);
```

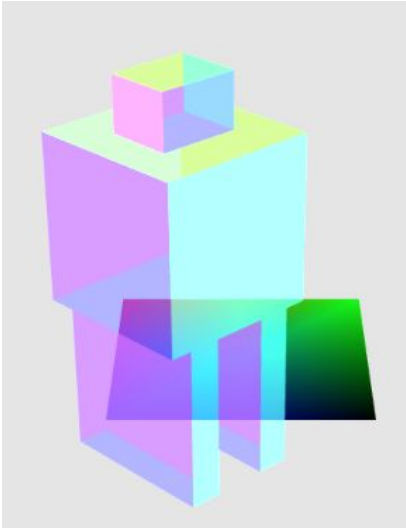
main.js

```
//need to specify how "precise" float should be
precision mediump float;

//interpolate argument between vertex and fragment shader
varying vec3 v_color;

//alpha value determining transparency
uniform float u_alpha;

//entry point again
void main() {
  //gl_FragColor ... magic output variable containg the final 4D color of the fragment

  //in our case we use the provided interpolated color from our three vertices
  //TASK 1-4
  gl_FragColor = vec4(v_color, u_alpha);
}
```

simple.fs.glsl

Blending

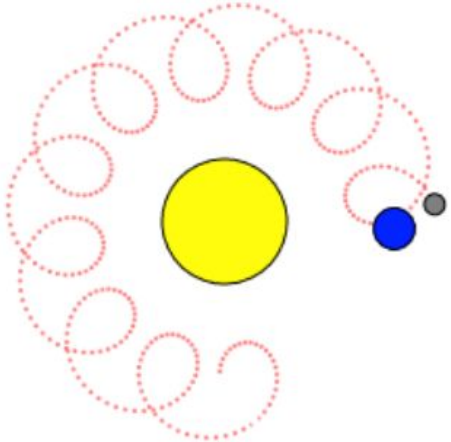
Keep in mind: The rendering order is important!
Depending on the viewing perspective some faces of the head are rendered in a wrong order
In practice: sort objects by depth and render from back to front

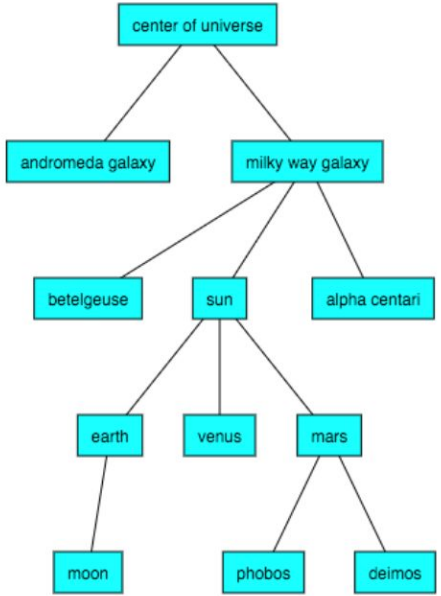


Scene Graphs

Tree-like structure (hierarchical structure) for organizing scene
Objects that should appear in the scene will be added to the graph
During rendering, the graph will be traversed recursively
Advantages:

- Propagation of properties
- Unique point of access
- Reusable components
- Abstraction





http://webglfundamentals.org/webgl/lessons/webgl-scene-graph.html

Scene Graphs: Abstraction into Nodes

JOHANNES KEPLER UNIVERSITY LINZ
Institute of Computer Graphics

Entities in graph are called nodes

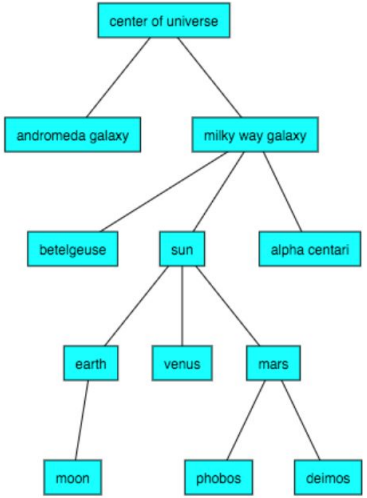
- Parent-child relationship of nodes

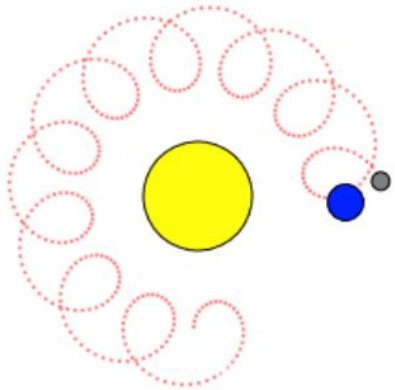
Properties of a node are applied to all children

- e.g., rotating the root node will make all sub nodes rotate too
- In the case of transformations, each node holds a matrix and all matrices of all sub nodes are multiplied by this matrix

Different types of nodes

- Geometry nodes (sphere, models, etc.)
- Transformation nodes
- Shader nodes (for lighting, materials, etc.)
- Today we will implement all three!





Scene Graph Traversal

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

Traversal works recursively

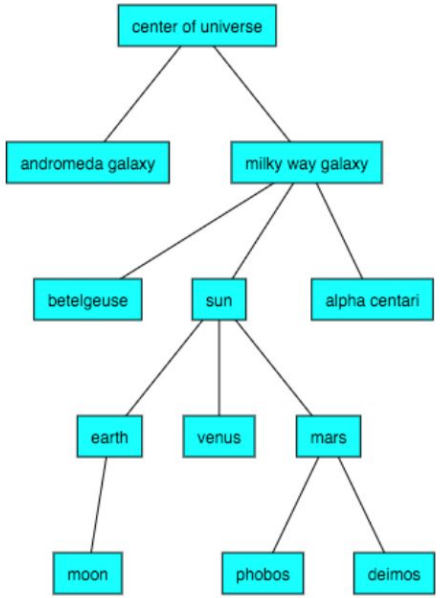
```
worldMatrix = parent(worldMatrix) * self(localMatrix)
```

Matrices are multiplied from top to bottom

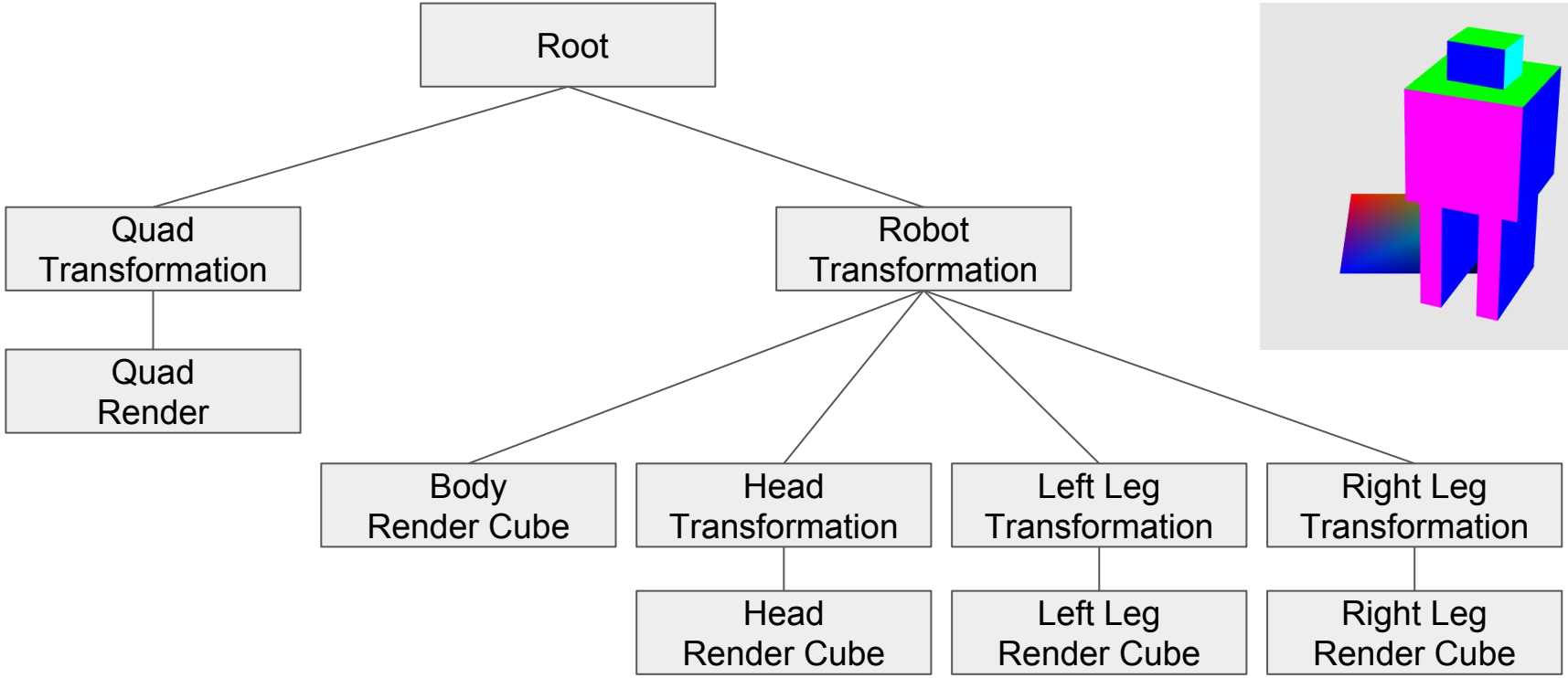
```
worldMatrix = greatGrandParent * grandParent * parent *
self(localMatrix)
```

Solar system example:

```
worldMatrixForMoon = galaxyMatrix * starMatrix *
planetMatrix * moonMatrix;
```



Let’s Make a Scene Graph for our Robot



Creating a Node

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

We need the functionality to:

- Append a child node
- Remove a child node
- Render node and all its children (recursively)

What we also need: Render context

- Stores scene matrix, view matrix, projection matrix and shader
- Gives us access to shader and matrices from inside nodes

Render Context

```
const context = createSceneGraphContext(gl, shaderProgram);
```

```
/**
 * returns a new rendering context
 * @param gl the gl context
 * @param projectionMatrix optional projection Matrix
 * @returns {ISceneGraphContext}
 */
function createSceneGraphContext(gl, shader) {

  //create a default projection matrix
  projectionMatrix = mat4.perspective(mat4.create(), fieldOfViewInRadians, aspectRatio, 0.01, 10);
  gl.uniformMatrix4fv(gl.getUniformLocation(shader, 'u_projection'), false, projectionMatrix);

  return {
    gl: gl,
    sceneMatrix: mat4.create(),
    viewMatrix: calculateViewMatrix(),
    projectionMatrix: projectionMatrix,
    shader: shader
  };
}
```

main.js

Base Node

```
//create scenegraph
rootNode = new SceneGraphNode();
```

Root

```
/**
 * base node of the scenegraph
 */
class SceneGraphNode {

  constructor() {
    this.children = [];
  }

  /**
   * appends a new child to this node
   * @param child the child to append
   * @returns {SceneGraphNode} the child
   */
  append(child) {
    this.children.push(child);
    return child;
  }

  /**
   * removes a child from this node
   * @param child
   * @returns {boolean} whether the operation was successful
   */
  remove(child) {
    var i = this.children.indexOf(child);
    if (i >= 0) {
      this.children.splice(i, 1);
    }
    return i >= 0;
  };

  /**
   * render method to render this scengraph
   * @param context
   */
  render(context) {

    //render all children
    this.children.forEach(function (c) {
      return c.render(context);
    });
  };
}
```

main.js

Task 2: Create a Quad Render Node

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

Goal: Implement empty `QuadRenderNode` template

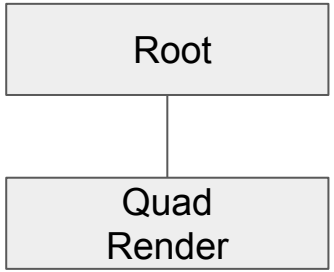
Step 0: Comment out `renderQuad` and `renderRobot` calls

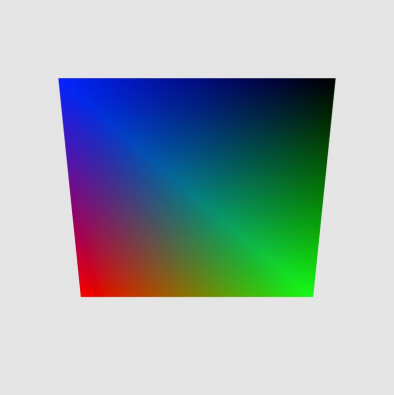
Step 1: Move quad render code to `QuadRenderNode` class

- Note: Take `sceneMatrix` and `viewMatrix` from render context (`context`)
- Note: Without transformations!

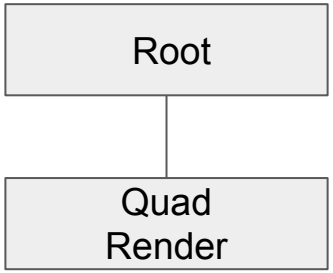
Step 2: Create node and add it to scene graph

- Note: Cube looks distorted because of perspective projection and camera





Task 2: Solution




```
//TASK 2-2
var quadNode = new QuadRenderNode();
rootNode.append(quadNode);
```

main.js

```
/**
 * a quad node that renders floor plane
 */
class QuadRenderNode extends SceneGraphNode {

  render(context) {

    //TASK 2-1

    //setting the model view and projection matrix on shader
    setUpModelViewMatrix(context.sceneMatrix, context.viewMatrix);

    var positionLocation = gl.getAttribLocation(context.shader, 'a_position');
    gl.bindBuffer(gl.ARRAY_BUFFER, quadVertexBuffer);
    gl.vertexAttribPointer(positionLocation, 2, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(positionLocation);

    var colorLocation = gl.getAttribLocation(context.shader, 'a_color');
    gl.bindBuffer(gl.ARRAY_BUFFER, quadColorBuffer);
    gl.vertexAttribPointer(colorLocation, 4, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(colorLocation);

    //set alpha value for blending
    //TASK 1-3
    gl.uniform1f(gl.getUniformLocation(context.shader, 'u_alpha'), 1);

    // draw the bound data as 6 vertices = 2 triangles starting at index 0
    gl.drawArrays(gl.TRIANGLES, 0, 6);

    //render children
    super.render(context);
  }
}
```

main.js

Transformation Node

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

Stores transformation matrix

During rendering:

1. Backup current `sceneMatrix` from context
2. Multiply `sceneMatrix` with local matrix
3. Render children
4. Restore previous `sceneMatrix`

```
//TASK 3-0
/**
 * a transformation node, i.e applied a transformation matrix to its successors
 */
class TransformationSceneGraphNode extends SceneGraphNode {
  /**
   * the matrix to apply
   * @param matrix
   */
  constructor(matrix) {
    super();
    this.matrix = matrix || mat4.create();
  }

  render(context) {
    //backup previous one
    var previous = context.sceneMatrix;
    //set current world matrix by multiplying it
    if (previous === null) {
      context.sceneMatrix = mat4.clone(this.matrix);
    }
    else {
      context.sceneMatrix = mat4.multiply(mat4.create(), previous, this.matrix);
    }

    //render children
    super.render(context);
    //restore backup
    context.sceneMatrix = previous;
  }

  setMatrix(matrix) {
    this.matrix = matrix;
  }
}
```

main.js

Task 3: Create a Transformation Node

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

Goal: Apply quad transformations

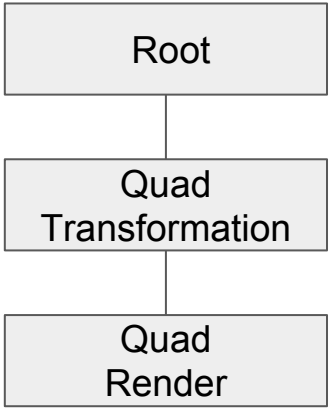
Step 0: Take a look at `TransformationSceneGraphNode`

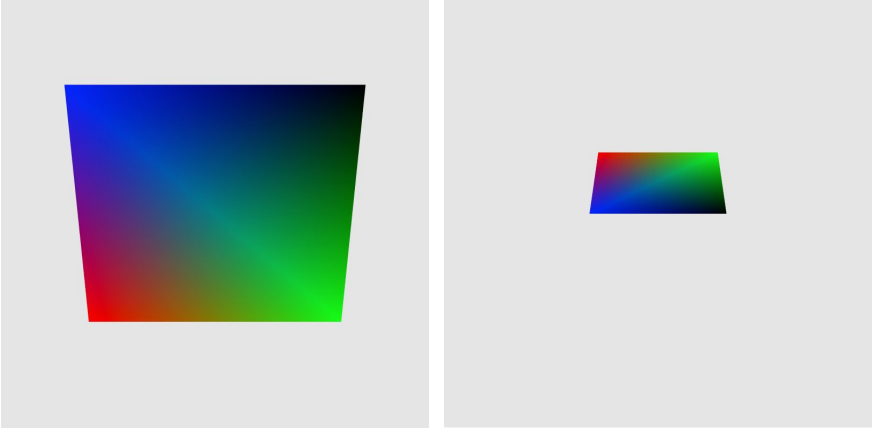
Step 1: Create local transformation matrix

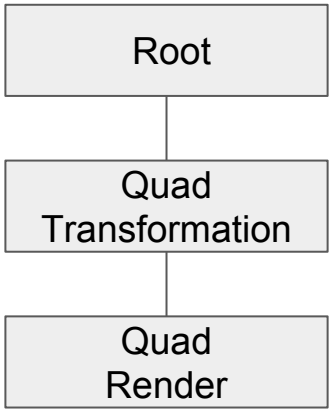
Move transformation code from `renderQuad`

Step 2: Create transformation node and add to scene graph

Hint: Don't forget to update existing scene graph parent-child relationships!







Task 3: Solution

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

```
//Task 3-1
var quadTransformationMatrix = glm.rotateX(90);
quadTransformationMatrix = mat4.multiply(mat4.create(), quadTransformationMatrix, glm.translate(0.0,-0.5,0));
quadTransformationMatrix = mat4.multiply(mat4.create(), quadTransformationMatrix, glm.scale(0.5,0.5,1));

//Task 3-2
var transformationNode = new TransformationSceneGraphNode(quadTransformationMatrix);
rootNode.append(transformationNode);

//TASK 2-2
var quadNode = new QuadRenderNode();
transformationNode.append(quadNode);
```

main.js

Task 4: Create a Cube Render Node

JOHANNES KEPLER UNIVERSITY LINZ

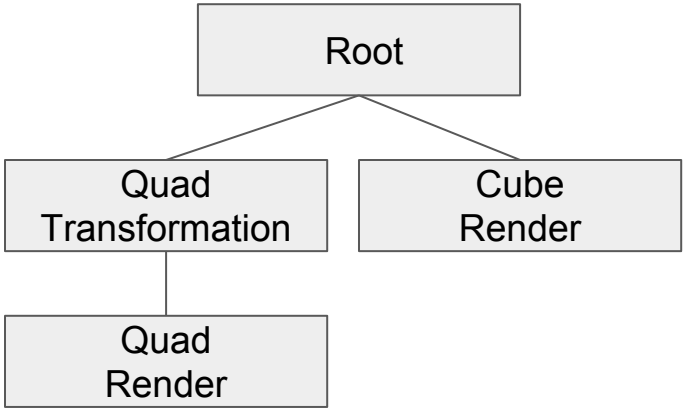
Institute of Computer Graphics

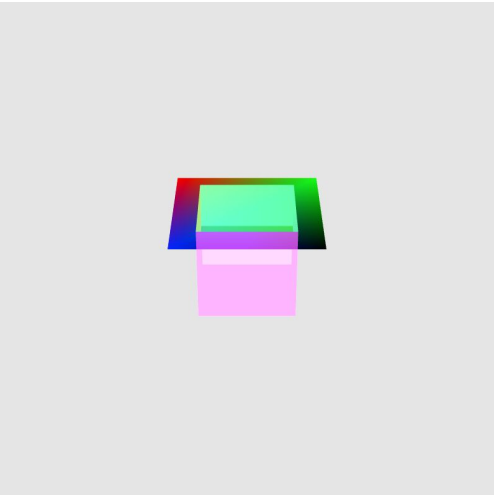
Goal: Render a cube

Step 1: Implement `CubeRenderNode`

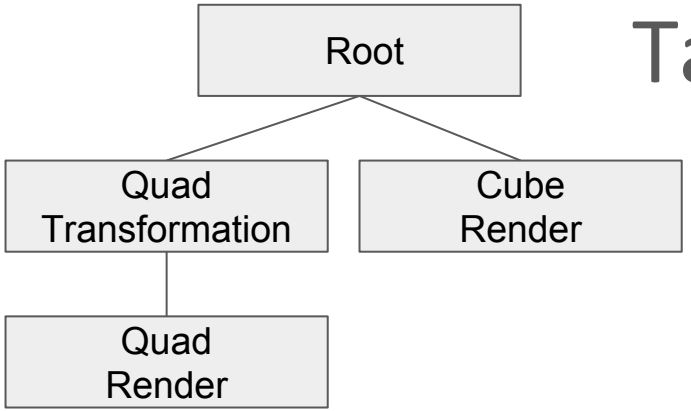
- Take `QuadRenderNode` as template
- Use cube render code from `renderRobot`

Step 2: Create cube node and add as child of root node





Task 4: Solution




```
//TASK 4-2
var cubeNode = new CubeRenderNode();
rootNode.append(cubeNode);
```

```
//TASK 4-1
/**
 * a cube node that renders a cube at its local origin
 */
class CubeRenderNode extends SceneGraphNode {

  render(context) {

    //setting the model view and projection matrix on shader
    setUpModelViewMatrix(context.sceneMatrix, context.viewMatrix);

    var positionLocation = gl.getAttribLocation(context.shader, 'a_position');
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexBuffer);
    gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false,0,0) ;
    gl.enableVertexAttribArray(positionLocation);

    var colorLocation = gl.getAttribLocation(context.shader, 'a_color');
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeColorBuffer);
    gl.vertexAttribPointer(colorLocation, 3, gl.FLOAT, false,0,0) ;
    gl.enableVertexAttribArray(colorLocation);

    //set alpha value for blending
    //TASK 1-3
    gl.uniform1f(gl.getUniformLocation(context.shader, 'u_alpha'), 0.5);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeIndexBuffer);
    gl.drawElements(gl.TRIANGLES, cubeIndices.length, gl.UNSIGNED_SHORT, 0); /

    //render children
    super.render(context);
  }
}
```

main.js

Shader Node

Holds shader program as variable

Allows us to use multiple shaders

During rendering:

1. Backup current `shader` from context
2. Use program (and set projection)
3. Render children
4. Restore previous `shader`

```
//TASK 5-0
/**
 * a shader node sets a specific shader for the successors
 */
class ShaderSceneGraphNode extends SceneGraphNode {
  /**
   * constructs a new shader node with the given shader program
   * @param shader the shader program to use
   */
  constructor(shader) {
    super();
    this.shader = shader;
  }

  render(context) {
    //backup prevoius one
    var backup = context.shader;
    //set current shader
    context.shader = this.shader;
    //activate the shader
    context.gl.useProgram(this.shader);
    //set projection matrix
    gl.uniformMatrix4fv(gl.getUniformLocation(context.shader, 'u_projection'),
      false, context.projectionMatrix);
    //render children
    super.render(context);
    //restore backup
    context.shader = backup;
    //activate the shader
    context.gl.useProgram(backup);
  }
};
```

main.js

Task 5: Adding a Shader Node

Goal: Assign yellow color to floor quad

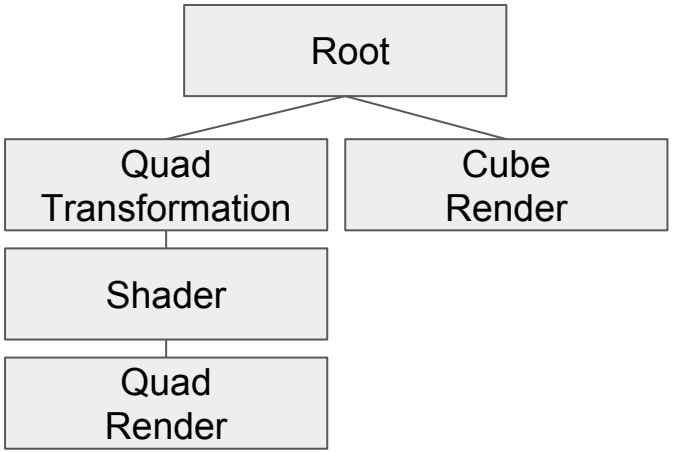
Step 1: Duplicate `simple.vs.glsl` shader and rename it

Step 2: Modify shader to apply static yellow color (R:1, G:1, B:0)

Step 3: Load new vertex shader as resource in `loadResources`

Step 4: Create new shader node and add it before the quad node

Note: create new shader program and provide it via constructor to shader node

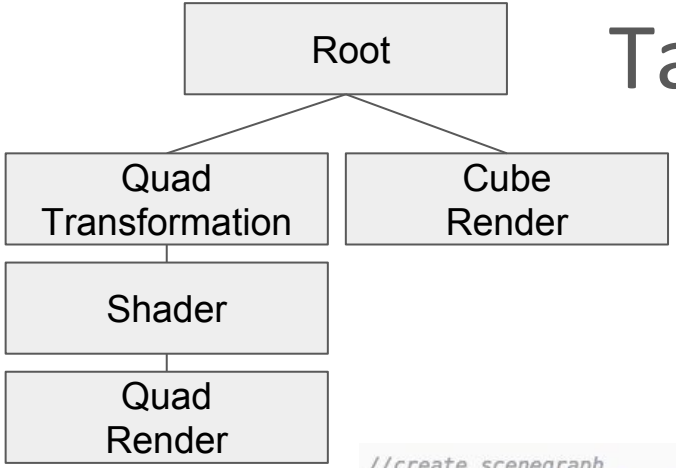


Task 5-1 and 5-2: Solution

```
//like a C program main is the main function
void main() {

  gl_Position = u_projection * u_modelView
    * vec4(a_position, 1);

  //TASK 5-2
  //we don't use a_color anymore
  a_color;

  //setting a static color (yellow) to the output varying color
  v_color = vec3(1,1,0);
}
```

static_color.vs.glsl

Task 5-3 and 5-4: Solution




```
//load the shader resources using a utility function
loadResources({
  vs: 'shader/simple.vs.glsl',
  fs: 'shader/simple.fs.glsl',
  //TASK 5-3
  staticcolorvs: 'shader/static_color.vs.glsl'
}).then(function (resources /*an object containing our keys with the loaded resources*/) {
  init(resources);

  //render one frame
  render();
});
```

main.js

```
//create scenegraph
rootNode = new SceneGraphNode();

//Task 3-1
var quadTransformationMatrix = glm.rotateX(90);
quadTransformationMatrix = mat4.multiply(mat4.create(), quadTransformationMatrix, glm.translate(0.0,-0.5,0));
quadTransformationMatrix = mat4.multiply(mat4.create(), quadTransformationMatrix, glm.scale(0.5,0.5,1));

//Task 3-2
var transformationNode = new TransformationSceneGraphNode(quadTransformationMatrix);
rootNode.append(transformationNode);

//TASK 5-3
var staticColorShaderNode = new ShaderSceneGraphNode(createProgram(gl, resources.staticcolorvs, resources.fs));
transformationNode.append(staticColorShaderNode);

//TASK 2-2
var quadNode = new QuadRenderNode();
staticColorShaderNode.append(quadNode);
```

main.js

Reference Scene Graph Implementation

JOHANNES KEPLER UNIVERSITY LINZ
Institute of Computer Graphics

Frameworks typically provide a scene graph implementation

Example: Three.js

Our framework provides the following nodes:

- `SGNode` (base class)
- `TransformationSGNode`
- `ShaderSGNode`
- `RenderSGNode`

From now on, we will only use the framework's scene graph impl.

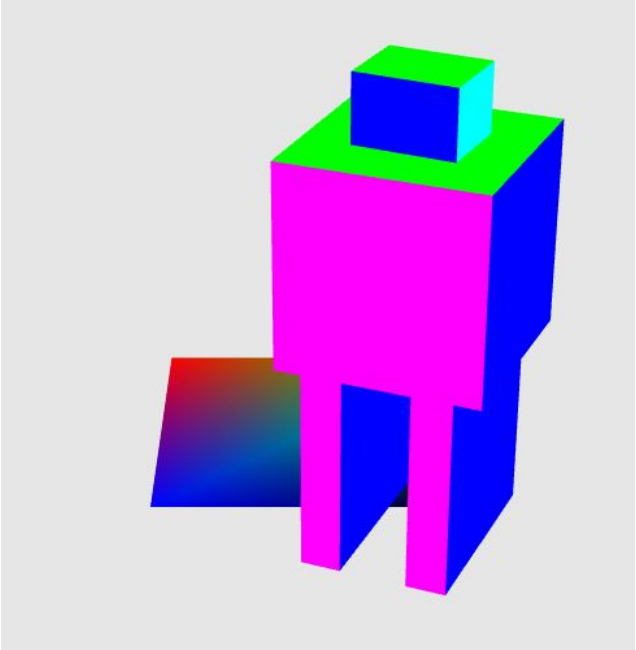
Recap

Depth Handling

Blending

Scene Graphs

- Abstraction into Nodes
- Scene graph traversal
- Base node class
- Render nodes
- Transformation nodes
- Implement robot using a scene graph



Useful Resource: http://webglfundamentals.org

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

WebGL Fundamentals

English

WebGL from the ground up. No magic.

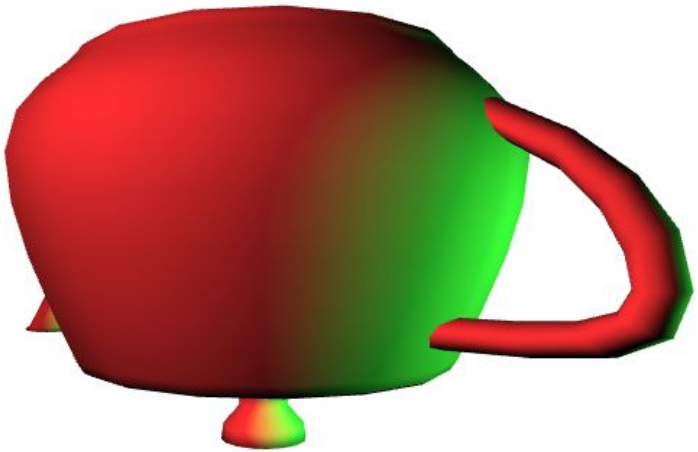
These are a set of articles that teach WebGL from basic principles. They are NOT old rehashed out of date OpenGL articles like many others on the net. They are entirely new, discarding the old out of date ideas and bringing you to a full understanding of what WebGL really is and how it really works.

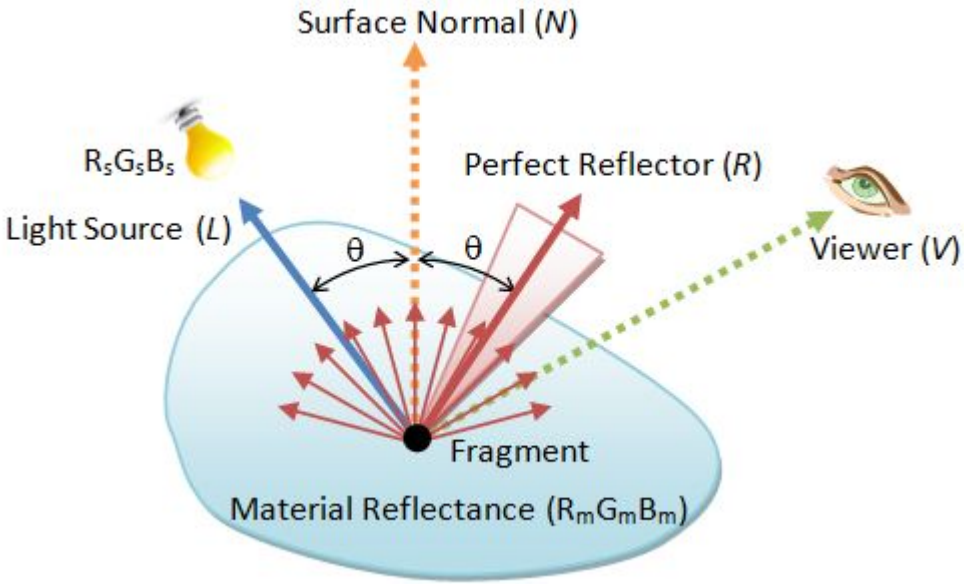
- Fundamentals
  - WebGL Fundamentals
  - WebGL How It Works
  - WebGL Shaders and GLSL
- Image Processing
  - WebGL Image Processing
  - WebGL Image Processing Continued
- 2D translation, rotation, scale, matrix math
  - WebGL 2D Translation
  - WebGL 2D Rotation
  - WebGL 2D Scale
  - WebGL 2D Matrices
- 3D
  - WebGL - Orthographic 3D
  - WebGL 3D Perspective
  - WebGL 3D - Cameras
- Lighting
  - WebGL 3D - Directional Lighting
  - WebGL 3D - Point Lighting
- Structure and Organization
  - WebGL - Less Code, More Fun
  - WebGL - Drawing Multiple Things
  - WebGL - Scene Graphs

Next Time

Illumination and shading

- Light
- Material





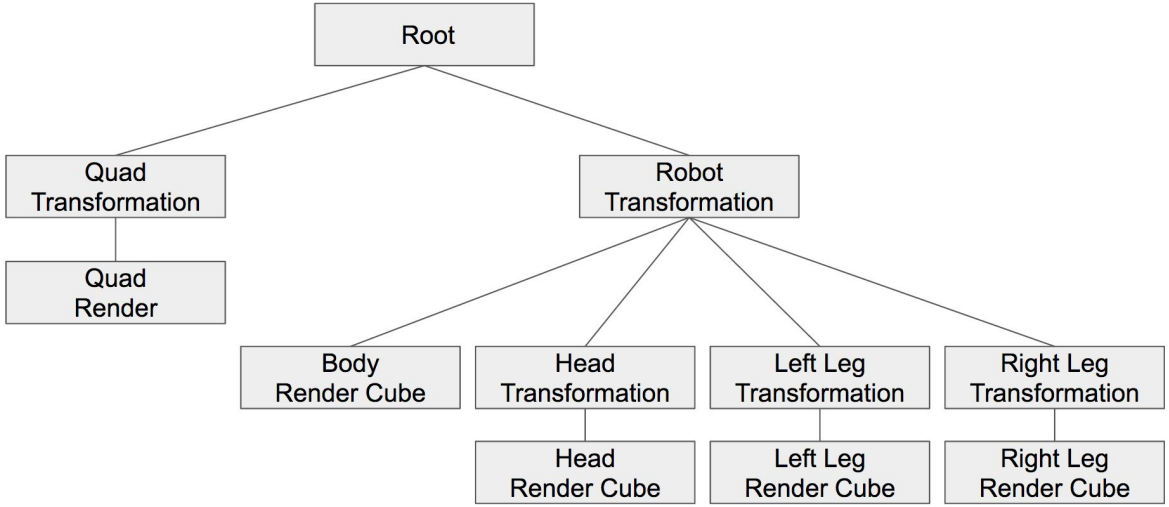
Play at Home!

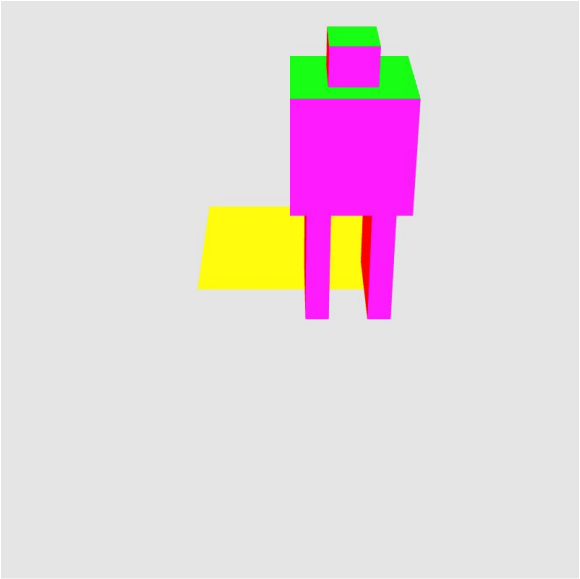
Task 6: Create Robot Scene Graph

Goal: Rebuild robot from last lab as a scene graph

Step 1: Construct a robot scene graph in `createRobot` function

Step 2: Update rotation in `render` function





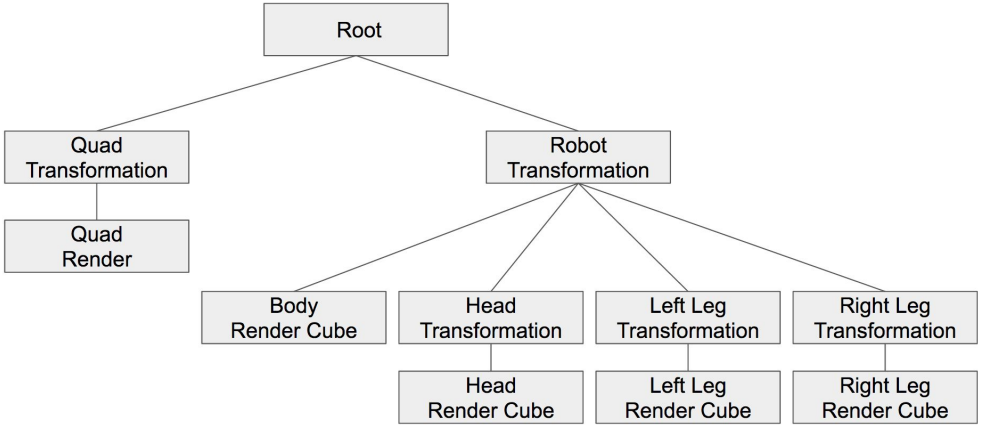
Task 6-1: Solution

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computer Graphics

```
function createRobot(rootNode) {

  //TASK 6

  //transformations of whole body
  var robotTransformationMatrix = mat4.multiply(mat4.create(), mat4.create(), glm.rotateY(animatedAngle/2));
  robotTransformationMatrix = mat4.multiply(mat4.create(), robotTransformationMatrix, glm.translate(0.3,0.9,0));
  robotTransformationNode = new TransformationSceneGraphNode(robotTransformationMatrix);
  rootNode.append(robotTransformationNode);

  //body
  cubeNode = new CubeRenderNode();
  robotTransformationNode.append(cubeNode);

  //transformation of head
  var headTransformationMatrix = mat4.multiply(mat4.create(), mat4.create(), glm.rotateY(animatedAngle));
  headTransformationMatrix = mat4.multiply(mat4.create(), headTransformationMatrix, glm.translate(0.0,0.4,0));
  headTransformationMatrix = mat4.multiply(mat4.create(), headTransformationMatrix, glm.scale(0.4,0.33,0.5));
  headTransformationNode = new TransformationSceneGraphNode(headTransformationMatrix);
  robotTransformationNode.append(headTransformationNode);

  //head
  cubeNode = new CubeRenderNode();
  headTransformationNode.append(cubeNode);

  //transformation of left leg
  var leftLegTransformationMatrix = mat4.multiply(mat4.create(), mat4.create(), glm.translate(0.16,-0.6,0));
  leftLegTransformationMatrix = mat4.multiply(mat4.create(), leftLegTransformationMatrix, glm.scale(0.2,1,1));
  var leftLegTransformationNode = new TransformationSceneGraphNode(leftLegTransformationMatrix);
  robotTransformationNode.append(leftLegTransformationNode);

  //left leg
  cubeNode = new CubeRenderNode();
  leftLegTransformationNode.append(cubeNode);

  //transformation of right leg
  var rightLegTransformationMatrix = mat4.multiply(mat4.create(), mat4.create(), glm.translate(-0.16,-0.6,0));
  rightLegTransformationMatrix = mat4.multiply(mat4.create(), rightLegTransformationMatrix, glm.scale(0.2,1,1));
  var rightLegtTransformationNode = new TransformationSceneGraphNode(rightLegTransformationMatrix);
  robotTransformationNode.append(rightLegtTransformationNode);

  //right leg
  cubeNode = new CubeRenderNode();
  rightLegtTransformationNode.append(cubeNode);
}
```

main.js



Task 6-2: Solution

```
//TASK 6-2
//update transformation of robot for rotation animation
var robotTransformationMatrix = mat4.multiply(mat4.create(), mat4.create(), glm.rotateY(animatedAngle/2));
robotTransformationMatrix = mat4.multiply(mat4.create(), robotTransformationMatrix, glm.translate(0.3,0.9,0));
robotTransformationNode.setMatrix(robotTransformationMatrix);

var headTransformationMatrix = mat4.multiply(mat4.create(), mat4.create(), glm.rotateY(animatedAngle));
headTransformationMatrix = mat4.multiply(mat4.create(), headTransformationMatrix, glm.translate(0.0,0.4,0));
headTransformationMatrix = mat4.multiply(mat4.create(), headTransformationMatrix, glm.scale(0.4,0.33,0.5));
headTransformationNode.setMatrix(headTransformationMatrix);
```

main.js - in render function