Computer Graphics

Programming III

Programming Assignment 2

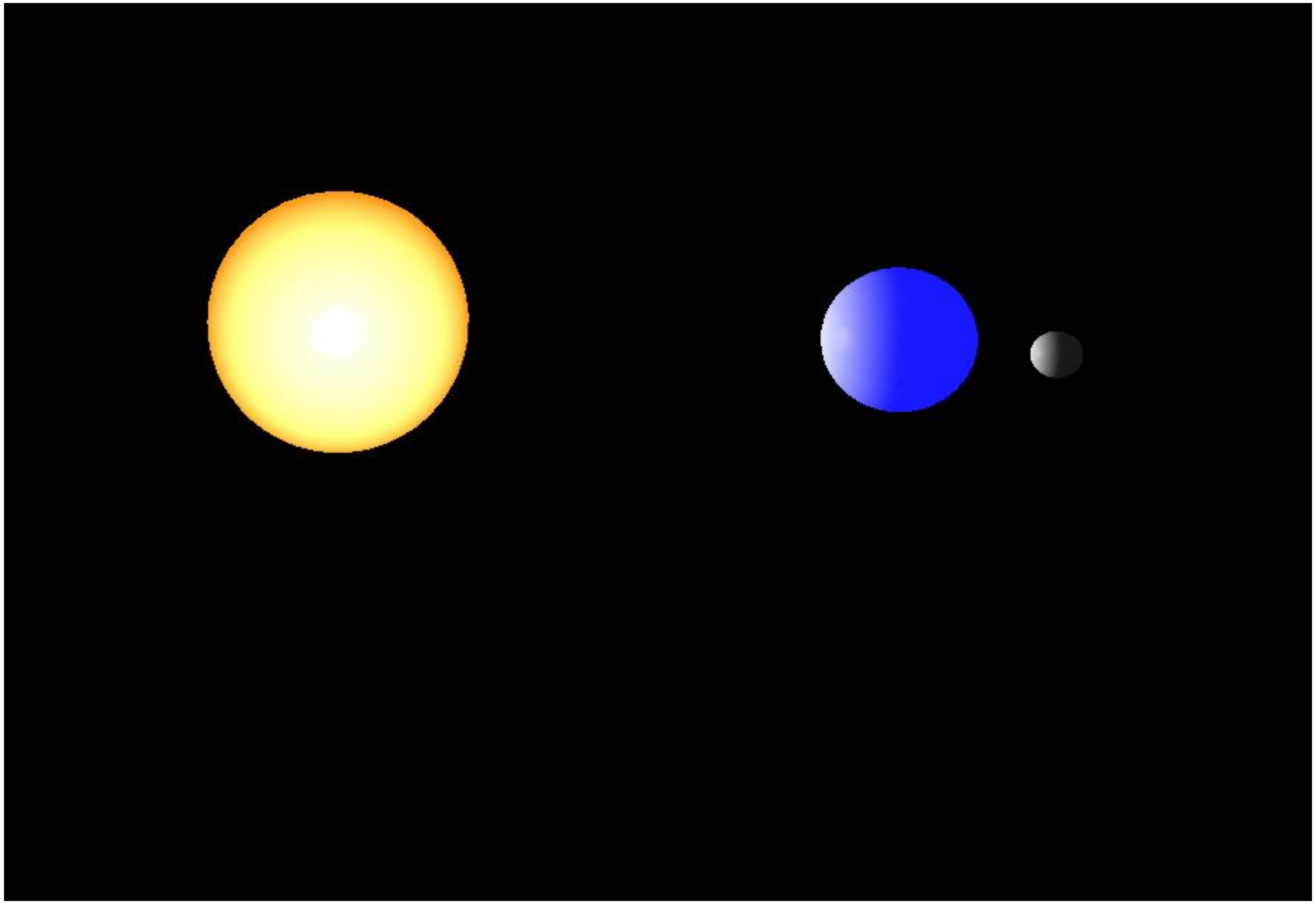
- Deadline today!
- Extension is only permitted with written request!

Programming Assignment 3

- Grading
  - Total points as 8, and 5 to pass the assignment.
- Contents
  - Hierarchical modeling, light shading, input output device control
- Deadline 2019-05-16 24:00

UNIVERSITY OF OULU

Animate a Solar System

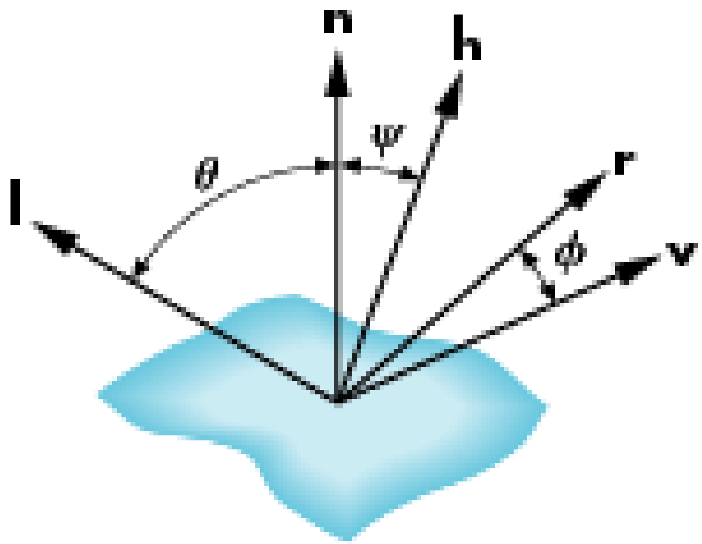


Animate a Solar System

- Render your own scene to animate a solar system with revolutions and rotations. In the scene there are three objects.
  – The sun. Rotate on its own.
  – The earth. Rotate on its own and rotate around the sun.
  – The moon. Rotate on its own and rotate around the earth.

Shading (Blinn-Phong model)

$$\boxed{I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^b + k_a I_a}$$



$$\boxed{\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|}$$

Shading calculations

- 3-D graphics V lecture has quite comprehensive theory on subject in general and about using Blinn-Phong model
  - Light properties (color, position, attenuation parameters, ...)
  - Material properties (material color, shininess, ...)
  - Object surface shape/position and normal vectors
    - Vertex positions determine distance, normal vectors relative angles

UNIVERSITY OF OULU

Available shading examples

- ExampleScene3
  - Scene renders a sphere with vertex colors
  - Vertex colors are calculated using Blinn-Phong shading in world/model coordinates and then just rendered as vertex colors
  - Calculations here assume that world coordinates match model coordinates and the object does not move.
  - Due to simplifications, implementation might be a bit more easy to understand

UNIVERSITY OF OULU

Available shading examples

- ExampleScene4
  - Scene renders a sphere with vertex colors
  - Calculations are now done in vertex shader
  - Initial vertex colors are ignored and constant light and material products are used instead
  - Shading colors are calculated using Blinn-Phong shading in view coordinates
    - View/Camera at origin

UNIVERSITY OF OULU

Using different material colors

- Example4 vertex shader uses constant "diffuseProduct" term (and similarly for ambient and specular components)

$$I = \underline{k_d \, I_d} \;\; \mathbf{l} \cdot \mathbf{n} \; + k_s \, I_s \, (\mathbf{n} \cdot \mathbf{h})^b + k_a \, I_a$$

- Very simple to calculate from different base material colors:
  - vec4 diffuseProduct = materialDiffuse * lightDiffuse;
  - materialDiffuse can come from vertex colors, uniform color variable, texture color

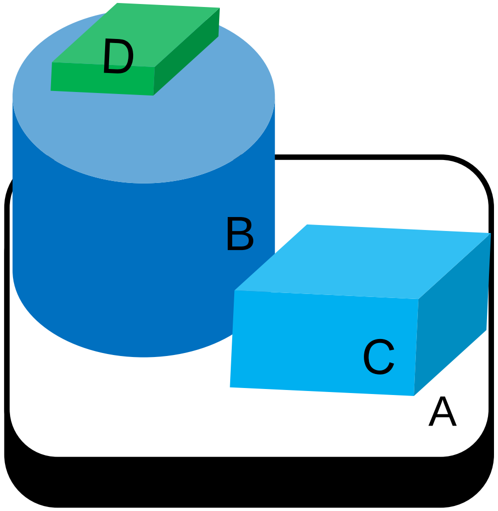
Blinn Phong Shading

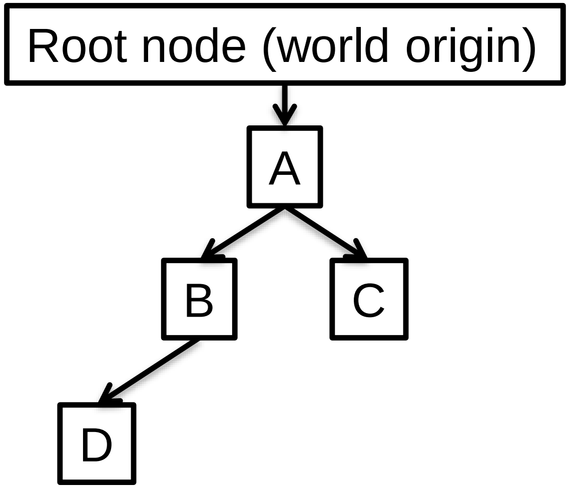
- Implement shading in fragment shader!
  - The best looking results as well
- Vertex shader should pass needed information to fragment shader
  - Some can be uniforms in fragment shader

Scene graph

Scene graph

- General idea: **Scene graph organizes objects into a hierarchical structure where state of parent objects also affect their child objects but nothing else**





Scene graph

```
glm_ModelViewMatrix.push(glm::mat4());
glm_ModelViewMatrix.top() = viewMat*glm_ModelViewMatrix.top();

glm::vec4 lightInViewPos_sun = glm_ModelViewMatrix.top() * lightPos_sun;
glm::vec4 lightInViewPos_planet = glm_ModelViewMatrix.top() * lightPos;

glUseProgram(shaderProgram_p1.getShaderProgram());
glm_ModelViewMatrix.top() = glm::translate(glm_ModelViewMatrix.top(), glm::vec3(lightPos_sun_x/ rwic
glm_ModelViewMatrix.top() = glm::rotate(glm_ModelViewMatrix.top(), rotation_p1, glm::vec3(0.0, 1.0,
glm_ModelViewMatrix.top() = glm::scale(glm_ModelViewMatrix.top(), glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(glGetUniformLocation(shaderProgram_p1.getShaderProgram(), "mvmatrix"), 1, GL_FALS
glUniformMatrix4fv(glGetUniformLocation(shaderProgram_p1.getShaderProgram(), "pmatrix"), 1, GL_FALSE
glUniform4fv(glGetUniformLocation(shaderProgram_p1.getShaderProgram(), "lightPosition"), 1, glm::val
glUniform4fv(glGetUniformLocation(shaderProgram_p1.getShaderProgram(), "color_p1"), 1, &color_p1[0])
glDrawElements(GL_TRIANGLES, static_cast<GLsizei>(sphereIndices_p1.size()), GL_UNSIGNED_SHORT, 0);


glm_ModelViewMatrix.push(glm_ModelViewMatrix.top());
glUseProgram(shaderProgram_p2.getShaderProgram());
glm_ModelViewMatrix.top() = glm::translate(glm_ModelViewMatrix.top(), glm::vec3(-4.0, 0.0, 0.0));
```

Dragging Mouse With SDL

```
    //Mouse button pressed
case SDL_MOUSEBUTTONDOWN:
    // See https://wiki.libsdl.org/SDL_MouseButtonEvent
    // Note: Mouse wheel has its own event
    {std::cout << "Mouse button down at : " << e.button.x << ", "
    switch (e.button.button)
    {
        case SDL_BUTTON_LEFT:
            mouse_down_x = e.button.x;
            mouse_down_y = e.button.y;
            break;

        default:
            std::cout << "Unknown (" << e.button.button << ")";
    }
    }
    break;
// Mouse button released
case SDL_MOUSEBUTTONUP:
    // See https://wiki.libsdl.org/SDL_MouseButtonEvent
    mouse_up_x = e.button.x;
    mouse_up_y = e.button.y;
    actionchange = true;
```

Suggestions for programming assignment 3

- End result will likely be simpler to implement by creating single more complex shader program
  - First get one of them working and then add the other part

- If have trouble with getting your shader to work
  - Some value just seems incorrect on the screen..
  - You can often implement parts of those calculations in your C++ program and print out the results
  - Problems are much easier to debug on CPU side than on GPU
  - Easy to port implementations due to using GLM for math