Exam – Solutions
EDA221 Computer Graphics : Introduction to 3D

2012–10–22

Note that this document only shows suggested solutions, and may not represent the *exact* solutions needed to get full score on the exam.

1. Shading

   a) Shader example:
   It sets the color of the pixel to blue. When the shader is applied to a primitive, all visible pixels covered by the primitives will be colored blue.

   b) Phong shading, and its three terms.

   $$I = \sum_i \left(k_a L_{i_a} + k_d \max(0, \mathbf{l}_i \cdot \mathbf{n}) L_{i_d} + k_s \max(0, (\mathbf{r}_i \cdot \mathbf{v})^\alpha) L_{i_s}\right)$$

   An ambient, diffuse and specular term. Replacing the specular term with Blinn-Phong specular $(\mathbf{n} \cdot \mathbf{h})^\alpha$ is also an accepted solution. For full point, define each term, the coefficients and the included vectors, notice that there can be several light sources and that the vectors need to be normalized.

   c) White paper: pure diffuse material. $k_a = (0, 0, 0)$ (this term approximates indirect lighting and should be set low or zero, in any case, it should be considerably less than the diffuse term), $k_d = (.7, .7, .7)$ and $k_s = (0, 0, 0)$.

   d) Red, shiny, plastic: The important thing is that the material contains a specular term and a non-zero value for the shininess coefficient and a red diffuse color. $k_a = (0, 0, 0)$ (this term approximates indirect lighting and should be set low or zero), $k_d = (0.7, 0, 0)$ and $k_s = (0.3, 0.3, 0.3)$.

   e) Gouraud versus Phong shading executed per-pixel. In Gouraud shading, the shaded color is computed at the vertex level and is then interpolated over the triangle surface in the rasterization stage. Therefore, small features in the shader, such as specular highlights or high frequency textures, will be smeared out. The diffuse term often looks fine however if it doesn't vary much spatially (e.g., k_d is read from a texture). In per-pixel Phong, the varying inputs to the shader (i.e., the vertex atributes such as the normal), are interpolated for each pixel, and the full Phong model is executed per-pixel using the smoothly interpolated normal at each execution, which gives high quality specular highlights.

2. Transforms
   Given the three matrices **A**: translation along the vector $\mathbf{v} = (4, 0, 2)$, **B**: rotation 90 degrees around the z-axis and **C**: a non-uniform scaling with 2 in x, 3 in y and 4 in z.

   a) Give the (4×4) matrix form of each of **A**, **B** and **C**.

   $$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

   .

$$\mathbf{B} = \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & 0 & 0 \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

.

$$\mathbf{C} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

.

b) Given the point $P = (1, 2, 3, 1)$, $P' = \mathbf{CAB}P = (4, 3, 20, 1)$.

c) Give an example of two (different) transform matrices $\mathbf{M}$ and $\mathbf{N}$ such that $\mathbf{MN} = \mathbf{NM}$:
For example a scaling in x and a translation in y, two rotations around the same axis, two scaling matrices, or two translations.

d) Describe the transform needed to transform the triangle from A to B in Figure 1,

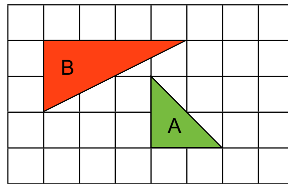

Figure 1: Transform from A to B

One possible solution:
Assume the origin is placed in the lower left corner of triangle A, and that Figure 1 shows the xy-plane. First, scale the triangle at A with two in y. Then, rotate the triangle 270 degrees (or minus 90 degrees) around the origin (around the z-axis, i.e., using the $\mathbf{R}_z$ rotation matrix). Finally, transform with the vector (-3,3,0). The matrix applied to the triangle at A is then:

$$\mathbf{T}(-3, 3, 0)\mathbf{R}_z(270)\mathbf{S}(1, 2, 1) \tag{4}$$

The full matrix is given by:

$$\mathbf{T}(-3, 3, 0)\mathbf{R}_z(270)\mathbf{S}(1, 2, 1) = \begin{bmatrix} 0 & 2 & 0 & -3 \\ -1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

In RenderChimp, each node has a **TRS** transform matrix, where the matrices are applied in that order. For this example, given that triangle A is defined with the three vertices $P_0 = (0, 0, 0, 1)$, $P_1 = (2, 0, 0, 1)$ and $P_2 = (0, 2, 0, 1)$, we can simply write:

```
tri->setScale(1,2,1);
tri->setRotateZ(fPI*3/2);
tri->setTranslate(-3, 3, 0);
```

Note that if we want to multiply the matrices together in *another* order, e.g., **SRT**, we must create new nodes in RenderChimp to handle this case.

3. Mapping techniques and GLSL

   a) Give three examples of how textures are used in shaders.
   1. Simple texture mapping, where the diffuse albedo (i.e., the k_d term) is fetched from a texture map, using the interpolated vertex attributes denoted *texture coordinates*. Similarly, the specular albedo (k_s) or the shininess factor (α) can be fetched from a texture.
   2. Cube mapping, where reflections (or refractions) are simulated by looking up into a texture using the view vector reflected (refracted) in the surface normal. In this use case, a special texture sampler is used, that uses a 3D direction to index into a so called cube map, which consists of six textures, approximating a spherical view of the environment.
   3. Bump mapping, where the perturbed normal is encoded as the RGB values of a texture. Here, the texture coordinated are used to index into the texture, and the resulting value is remapped from [0,1] to [-1,1]. The remapped value usually represents a normal perturbation in tangent space, i.e., the direction of the normal along the normal, tangent and binormal direction for the current surface point.

   b) What is tangent space and what is it useful for?
   Tangent space is a local orthonormal coordinate system for a parametric coordinate (u, v), where the basis vectors are the normalized surface tangent, normalized binormal and normalized normal at the parameter (u, v). The origin of the tangent space is the value of the surface at (u, v). Note that tangent space is unique for each point on a surface, unlike object space, which is constant for an entire model. Tangent space is useful for bump mapping, as it is a convenient coordinate space to express a perturbed normal vector. The three orthonormal basis vectors form a matrix that lets us transform between object space and tangent space.

   c) GLSL contains the function `reflect`. Give two examples when that function is useful.
   1. For computing the reflection vector $\mathbf{r}$ of the *light vector* $\mathbf{l}$ reflected in the surface normal which is used in the specular part of a phong shader. For example: $k_s \max(0, \mathbf{r} \cdot \mathbf{v})^\alpha$.
   2. For computing the reflection of the *view vector* in the surface normal when performing a cube map lookup. In this example, the reflection vector is used to index into a cube map, and the direction selects one of the six textures and the location therein.

   d) How do you specify a `float` input parameter which has the same value for all invocations of a shader in GLSL?
   With the `uniform` keyword. One example:

```
uniform float time;
```

   e) How do you pass a `vec3` color that has a unique value for each vertex, from a vertex shader to a pixel shader in GLSL?
   With the `in` and `out` keywords.
   Example: the variable `vec3 color` is passed from the vertex shader to the pixel shader:
   In the vertex shader:

```
in vec4 vPosition;
in vec3 vertexColor;
out vec3 color;
void main() {
    color = vertexColor;
    gl_Position = vPosition;
}
```

   In the pixel shader:

```
in vec3 color;
out vec4 fColor;
void main() {
    fColor = vec4(color,1);
}
```

   Note that in older versions of GLSL you use the `varying` keyword.

4. The Graphics Pipeline

a) Describe the responsibility of the vertex shader, rasterizer and pixel shader stage of the graphics pipeline.
The vertex shader works on vertices and transforms each vertex and its attributes (sent from the application through vertex arrays), such as the vertex normal, texture coordinates, etc. The vertex shader always outputs a position in clip space to the rasterizer (i.e., applies the ModelViewProjection matrix to each vertex).
The rasterizer computes visibility, i.e., in which pixels the current triangle is visible using inside tests based on *edge equations* It also interpolates the vertex attributes for each pixel. As part of the rasterization phase, the *depth buffer* test is executed to determine which of the triangles overlapping a pixel that is visible.
The pixel shaders receives the interpolated vertex attributes from the rasterizer and a set of uniforms as inputs and computes a color for the pixel. The pixel shader always outputs a color (unless the `discard` operation is called).

b) Mention three coordinate systems (spaces) that you may encounter in a rendering pipeline. Briefly explain the purpose of each system.
Object/Model space, which is a convenient space to describe/create each model in. This space is unique for each model.
World space, which is a reference frame where objects, lights and cameras are placed and positioned relative to each other.
View/Camera Space. A coordinate system with the camera at the origin. Usually looking along the negative z-axis (OpenGL convention).
Clip space. A coordinate system after the (ModelView)projection matrix has been applied. In this space, the (negative) z-value has been mapped to the w-coordinate.
Normalized Device Coordinates: The coordinate space we are in after the perspective divide has been performed on the clip space position, e.g.,$(x/w, y/w, z/w, 1)$.
Screen space. The 2D position on screen. This is obtained from NDC by a scale and offset.
Tangent space. See earlier question above.

c) What is backface culling, why is it useful and where in the graphics pipeline can a backface culling test be executed?
Backface culling removes triangles with face normals pointing away from the camera. The back face test can be defined as:
The triangle is back facing if:

$$\mathbf{n} \cdot \mathbf{p} > 0, \tag{6}$$

where $\mathbf{n}$ is the surface normal and $\mathbf{p}$ is one vertex of the triangle. Alternatively

$$\mathbf{n} \cdot \mathbf{v} < 0, \tag{7}$$

where $\mathbf{v}$ is the view vector at one vertex of the triangle (see Lecture 3, slide 38 for an illustration).
The backface test is typically performed after vertex shading as part of the rasterization stage when triangle has been transformed with the MVP matrix, just before visibility testing in the rasterizer. It can not be performed already in the vertex shader, as the VS works on each vertex individually, and cannot cull the entire triangle. In a scene, sometimes about 50% of the triangles are backfacing, so removing them can save a lot of work for the rasterizer and pixel shading stage.

d) A triangle has camera space vertex positions $P_0 = (1, 0, -1)$, $P_1 = (1, 1, -1)$ and $P_2 = (0, 1, -1)$, with normal pointing in the direction $(P_1 - P_0) \times (P_2 - P_0)$. Can this triangle be backface culled? Motivate your answer.
The surface normal point in direction: $\mathbf{n} = (0, 0, 1)$.

$$\mathbf{n} \cdot P_0 = (0, 0, 1) \cdot (1, 0, -1) = -1 < 0, \tag{8}$$

thus, the triangle can not be back face culled. In OpenGL, the camera looks along the negative z-axis, and in this case, this triangle normal points directly towards the camera.

5. Hierarchical Modeling
   In RenderChimp, the following geometry nodes are given:

```
world  = SceneGraph::createWorld(...);
sun    = SceneGraph::createGeometry(...);
earth  = SceneGraph::createGeometry(...);
```

   a) A scene graph is a hierarchical node-based structure and describes how objects are constructed from parts, and how the individual parts move relative to each other. It describes hierarchical transforms, such as the individual movements of the joints in an arm. Transforms can be applied locally to leaf nodes or on nodes higher up in the graph, such that a transform is applied to all nodes in the corresponding sub-graph.
   b) What is the difference between spin and orbit?
      With spin, an object simply rotates arounds its own axis. This can be performed by applying a rotation to the object in its local coordinate system.
      Orbit is a rotation around a fixed point, such a planet rotating around the sun. This can be performed by first applying a translation to the object away from the origin, then rotating the translated object around the origin, or *pivot* point.
   c) Show one example of spin and one example of orbit using the objects defined above. Add new nodes if necessary. Exact RenderChimp syntax is not required, but the structure of the scene graph should be clear from your solution.
      For a scene graph example illustration, see Seminar 1.
      For a possible RenderChimp implementation, see below or assignment 2.

```
void RCInit()
{
    world  = SceneGraph::createWorld(...);
    sun    = SceneGraph::createGeometry(...);
    earth  = SceneGraph::createGeometry(...);

    // Setup scene graph
    world->attachChild( sun );
    pivot = SceneGraph::createGroup("pivot");
    world->attachChild(pivot);
    earth->translate(dist_earth_to_sun, 0.0f, 0.0f);
    pivot->attachChild(earth);
}

void RCUpdate()
{
    float time = Platform::getFrameTimeStep();

    // ORBIT - earth around sun
    const float ang_vel = ...
    float orbit_rad = ang_vel * time;
    pivot->rotateY( orbit_rad );

    // SPIN - earth rotates around its axis
    const float ang_vel_spin = ...
    float spin_rad = ang_vel_spin * time
    earth->rotateY( ang_vel_spin );
}
```

6. General Computer Graphics
   a) How is color represented in computer graphics and how does this relate to the Human Visual System (HVS)?
      The eye contains the receptor types: rods and three types of cones. The three types of cones

are sensitive to colored lights. Rods are good for monochromatic light and night vision. In computer graphics, color is usually represented as an 8 bit value per color channel for each of R,G and B, or even an floating point value per channel for higher dynamic range and better color fidelity.

b) Mention three differences between between real-time graphics and offline (photorealistic) computer graphics. In this context, also explain why graphics hardware, e.g., graphics cards are useful for computer graphics.
RT: Time of one image: 20 ms, user interaction. Limited amount of detail/geometry. Approximations to get plausible image within the time budget. Simple shaders, lighting and animations.
PRCG: Photorealism. Time of one image: minutes or hours. High detail level. Very advanced shaders, lighting, and fine-tuned animation. Often no user interaction. The image is rendered once for use in animation or high quality still images, for product visualization, for example.
Another important difference is the underlying rendering algorithm. See the discussion in Lecture 6 about rasterization versus ray tracing. Rasterization cannot easily handle realistic reflections and refractions, indirect lighting etc. These effect are more easily expressed in a ray tracing engine, which more carefully approximates the rendering equation.
Graphics cards are designed to accelerate the most expensive parts of the rasterization graphics pipeline, such as vertex transforms, visibility computations (rasterization and depth buffering), barycentric interpolation and pixel shading. Thanks to wide parallel processors, graphics cards can render millions of triangles with moderately complex shading in real-time.

c) Why is the *rendering equation* so hard to solve?
It contains the light intensity on both the left and the right side, and must be recursively evaluated *for all points in the scene* for a correct solution. This is not tractable, and in practice, this equation is approximated or evaluated at a subset of points. Furthermore, determining mutual visibility between two points in the scene is a complex operation if the scene contains many objects. Also, the BRDF may be complex and hard to evaluate, and may vary for each point in the scene.

d) In the course, we have described linear and bilinear interpolation. There is also a concept called trilinear interpolation, that we haven't explicitly discussed. Based on the definition of linear and bilinear interpolation, how would you define trilinear interpolation? Motivate your definition.
Trilinear interpolation is applied to interpolate a value at any point $\mathbf{p}$ within a cube given values at the eight corners of the cube. Let $\mathbf{p}$ be described in a coordinate system of the cube so that $\mathbf{p}$ varies from [0,1] in each of x, y, and z as we move around in the cube. That is: $\mathbf{p} = (0,0,0)$ in the lower, left, frontmost corner of the cube, and $\mathbf{p} = (1,1,1)$ in the upper right, farthest corner. Now, pick two opposite faces of the cube, say the two faces parallel to the xy plane (the bottom and top face of the cube). Then perform bilinear interpolation using the xy-coordinate of $\mathbf{p}$ within each of these two faces to get two interpolated points inside each face. Finally, perform a linear interpolation using the z-coordinate of $\mathbf{p}$ between these two points. This process is called trilinear interpolation, and combines two bilinear interpolations in xy, with an linear interpolation in z.

The end.