VGP352 – Week 1

- Agenda:
  - Course Intro
  - Per-fragment lighting revisited
    - Phong Shading
    - Surface-space
  - Bump mapping
    - Basic usage
    - Bumpmap storage

8-April-2009

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes or portions of classes that conform to specific interfaces
- Graphics terminology and concepts
  - Polygon, pixel, texture, infinite light, point light, spot light, etc.
- Linear algebra and vector math
  - Matrix arithmetic

What should you already know?

- Material from VGP351:
  - Using OpenGL
    - Setting up shaders
    - Getting data in
    - etc.
  - Transformations
    - 3D space transformations
    - Projections
  - Lighting and shading
  - Texture mapping

What will you learn?

- Advanced lighting models
  - BRDFs
  - Fur and hair rendering
  - “Toon” and other non-photorealistic rendering

How will you be graded?

- Four bi-weekly quizzes
  - These are listed on the syllabus
- One final exam
- Three programming projects
  - The first will be pretty small...perhaps small enough to complete in class
  - The remaining two projects will be larger
- One in-class presentation

How will programs be graded?

- Does the program produce the correct output?
- Are appropriate algorithms and data-structures used?
- Is the code readable, clear, and properly documented?

How will the presentation be graded?

- During the term, several papers will be assigned to read
  - Select and present one of the assigned readings to the class
    - What is the problem being solved?
    - How does the paper's author solve that problem?
    - What is novel about the author's solution?
    - What questions does the paper leave unanswered?
  - Material from some papers may appear on bi-weekly quizzes

Class Web Site

- Syllabus, assignments, and base code:
  http://people.freedesktop.org/~idr/2009Q2-VGP352/

Phong Shading Recap

- Phong shading... aka per-fragment lighting
  - Calculate lighting parameters per-vertex
  - Interpolate calculated values
  - Calculate lighting per-fragment based on interpolated parameter values

Phong Shading Recap

```
attribute vec3 normal;
attribute vec4 color;
uniform mat3 normal_xform;
uniform mat4 vertex_xform;
uniform mat4 mvp;

varying vec3 vertex_normal;
varying vec4 vertex_color;
varying vec3 vertex;

void main(void)
{
    gl_Position = mvp * gl_Vertex;

    vertex_normal = normal_xform * normal;
    vertex_color = color;
    vertex = vertex_xform * gl_Vertex;
}
```

Phong Shading Recap

```
uniform vec3 eye_space_light;
varying vec3 vertex_normal;
varying vec4 vertex_color;
varying vec3 vertex;
const vec3 eye_space_eye = vec3(0);

void main(void)
{
    vec3 l = normalize(eye_space_light – vertex);
    vec3 v = normalize(eye_space_eye - vertex);
    vec3 h = normalize(l + v);
    float n_dot_l = dot(vertex_normal, l);
    vec4 diff = vertex_color * n_dot_l;
    float spec = pow(dot(n, h), 16.0);

    gl_FragColor = step(0.0, n_dot_l) *
        vec4(diff.xyz + vec3(spec), vertex_color.w);
}
```

Surface-Space

- From the point of view of the surface, what is the normal vector?
  - We'll call this *surface-space*

Surface-Space

- From the point of view of the surface, what is the normal vector?
  - We'll call this *surface-space*
  - Assuming the surface is flat, N_{surf} = (0, 0, 1)

Surface-Space

- If we know N_{world}, can we create transformation that will generate N_{surf}?
  - Not uniquely
    - An orthonormal basis requires three orthogonal, normalized vectors, but we only have one
    - This is the same reason we need the “up” vector to create the camera look-at transform
  - If only we had another vector in plane...

Surface-Space

- Create a new vector, and call it the *tangent*
  - Knowing N_{surf} and T_{surf} is enough to create an orthonormal basis
  - This basis can transform *any* vector to surface-space from object-space
    - N_{obj} is an obvious choice
    - For lighting, V and L need to be in the same space as N
- Because we use the tangent vector, surface-space is sometimes called *tangent-space*

Surface-Space

```
varying vec3 light_dir;
attribute vec3 tangent;
attribute vec3 normal;

void main(void)
{
    gl_Position = mvp * gl_Vertex;

    mat3 tbn = mat3(normal_xform * tangent,
                    normal_xform * normal,
                    cross(n, t));

    vec3 vert_pos = vec3(vertex_xform * gl_Vertex);
    vec3 light = eye_space_light - vert_pos;

    light_dir = normalize(light * tbn);
}
```

Surface-Space

```
varying vec3 light_dir;
attribute vec3 tangent;
attribute vec3 normal;

void main(void)
{
    gl_Position = mvp * gl_Vertex;

    mat3 tbn = mat3(normal_xform * tangent,
                    normal_xform * normal,
                    cross(n, t));

    vec3 vert_pos = vec3(vertex_xform * gl_Vertex);
    vec3 light = eye_space_light - vert_pos;

    light_dir = normalize(light * tbn);
}
```

This actually calculates M_s^T

Surface-Space

```
varying vec3 light_dir;
attribute vec3 tangent;
attribute vec3 normal;

void main(void)
{
    gl_Position = mvp * gl_Vertex;

    mat3 tbn = mat3(normal_xform * tangent,
                    normal_xform * normal,
                    cross(n, t));

    vec3 vert_pos = vec3(vertex_xform * gl_Vertex);
    vec3 light = eye_space_light - vert_pos;

    light_dir = normalize(light * tbn);
}
```

This actually calculates M_s^T

Remember: $Mv = vM^T$

Surface-Space

```
varying vec3 light_dir;
varying vec3 eye_dir;
varying vec4 vertex_color;

void main(void)
{
    vec3 l = normalize(light_dir);
    vec3 v = normalize(eye_dir);
    vec3 h = normalize(l + v);
    float n_dot_l = l.z;
    vec4 diff = vertex_color * n_dot_l;
    float spec = pow(h.z, 16.0);

    gl_FragColor = step(0.0, n_dot_l) *
        vec4(diff.xyz + vec3(spec), vertex_color.w);
}
```

Surface-Space

```
varying vec3 light_dir;
varying vec3 eye_dir;
varying vec4 vertex_color;

void main(void)
{
    vec3 l = normalize(light_dir);
    vec3 v = normalize(eye_dir);
    vec3 h = normalize(l + v);
    float n_dot_l = l.z;
    vec4 diff = vertex_color * n_dot_l;
    float spec = pow(h.z, 16.0);

    gl_FragColor = step(0.0, n_dot_l) *
        vec4(diff.xyz + vec3(spec), vertex_color.w);
}
```

Remember: N is (0, 0, 1)!

8-April-2009

Surface-Space

- What is B?
  - In the calculation: $B = N \times T$
  - Correctly, this is the bi-tangent
    - Many places incorrectly call it the bi-normal
    - Either way, we'll just call it B

Surface-Space

- What does this math headache gain us?
  - Just a trivial fragment shader optimization so far
    - Seems hardly worth it
  - What else?

Bump Mapping

- What if the surface isn't really flat or smoothly curved?
  - Just like few real surfaces have truly uniform color, few real surfaces have uniform normals
  - Use the same solution!
    - Store colors in an image → store normals in an image

Normal Map Storage

- Store the X, Y, and Z values of the surface-space normals in the R, G, and B components
  - Since Z tends to be close to 1.0, these images tend to look very blue

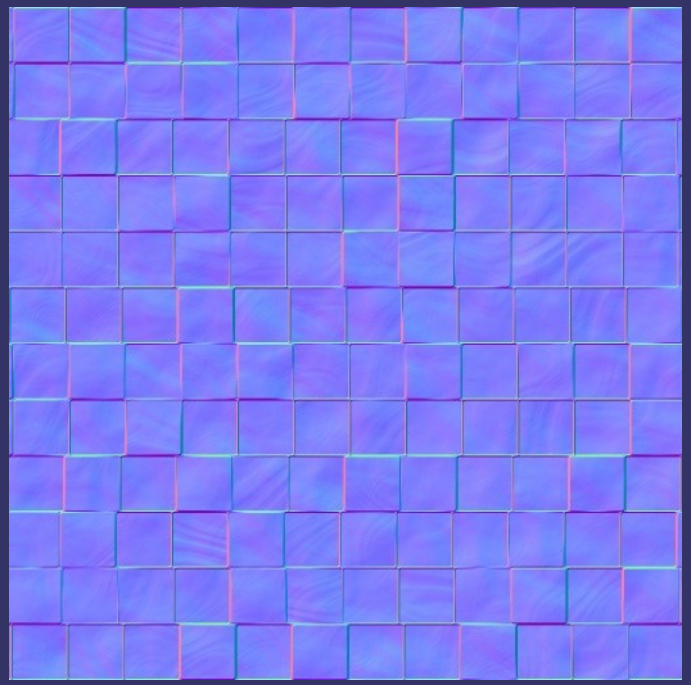

Image from http://www.filterforge.com/filters/243-normal.html

8-April-2009

Normal Map Storage

- What is the range of colors in a texture?

Normal Map Storage

- What is the range of colors in a texture?
  - [0.0, 1.0]
  - We have to convert these to the [-1, 1] range desired for normal directions
    - Just convert X and Y... Z must be > 0, so just leave it

Normal Map Storage

- We don't even need Z
  - Z must always be > 0.0
  - Derive it from X and Y:

Normal Map Storage

- We don't even need Z
  - Z must always be > 0.0
  - Derive it from X and Y:

$$\sqrt{x^2+y^2+z^2}=1.0$$
$$x^2+y^2+z^2=1.0$$
$$z^2=1.0-x^2-y^2$$
$$z=\sqrt{1.0-x^2-y^2}$$

Normal Map Storage

- 2-component textures can be achieved in a couple ways:
  - Use `GL_LUMINANCE_ALPHA`
    - Some hardware doesn't really support this, so it will silently convert it to RGBA...making it bigger
  - Use `GL_RG`
    - Requires `GL_ARB_texture_rg`
  - Use `GL_COMPRESSED_RED_GREEN_RGTC2_EXT`
    - Requires `GL_ARB_texture_compression_rgtc` or `GL_EXT_texture_compression_rgtc`
    - May add undesired compression artifacts

References

Lengyel, Eric. “Computing Tangent Space Basis Vectors for an Arbitrary Mesh”. Terathon Software 3D Graphics Library, 2001. http://www.terathon.com/code/tangent.html

Normal map photography tutorial:

http://www.zarria.net/nrmphoto/nrmphoto.html

OpenGL extension specs:

http://www.opengl.org/registry/specs/ARB/texture_rg.txt

http://www.opengl.org/registry/specs/ARB/texture_compression_rgtc.txt

Next week...

- Render-to-texture
- Environment mapping
  - Rendering to env maps
- Improving the reflection model
  - Using env maps as better lights
  - Fresnel reflection
- Read:

Michael Toksvig. “Mipmapping Normal Maps.”
http://developer.nvidia.com/object/mipmapping_normal_maps.html

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.

8-April-2009