Shader Applications

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Objectives

- Shader Applications
- Texture Mapping Applications
- Reflection Maps
- Bump Maps

Vertex Shader Applications

- Moving vertices
  - Morphing
  - Wave motion
  - Fractals
- Lighting
  - More realistic models
  - Cartoon shaders

Wave Motion Vertex Shader

```
uniform float time;
uniform float xs, zs, // frequencies
uniform float h; // height scale
uniform mat4 ModelView, Projection;
in vec4 vPosition;

void main() {
  vec4 t =vPosition;
  t.y = vPosition.y
     + h*sin(time + xs*vPosition.x)
     + h*sin(time + zs*vPosition.z);
  gl_Position = Projection*ModelView*t;
}
```

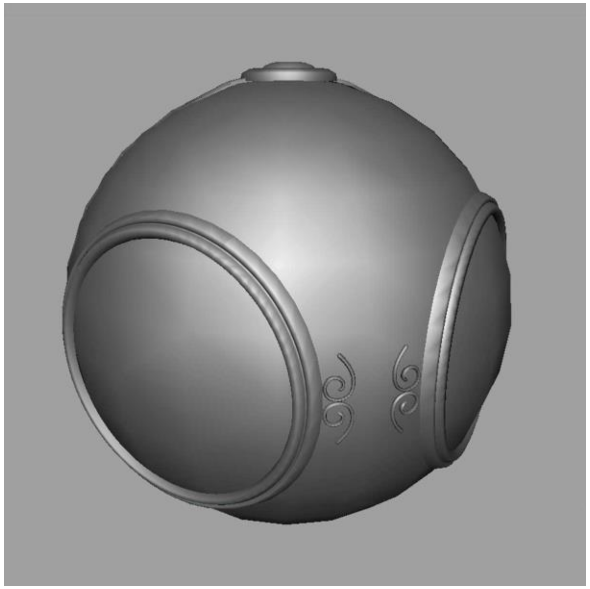
Particle System

```
uniform vec3 init_vel;
uniform float g, m, t;
uniform mat4 Projection, ModelView;
in vPosition;
void main(){
vec3 object_pos;
object_pos.x = vPosition.x + vel.x*t;
object_pos.y = vPosition.y + vel.y*t
       + g/(2.0*m)*t*t;
object_pos.z = vPosition.z + vel.z*t;
gl_Position = Projection*
   ModelView*vec4(object_pos,1);
}
```

The University of New Mexico

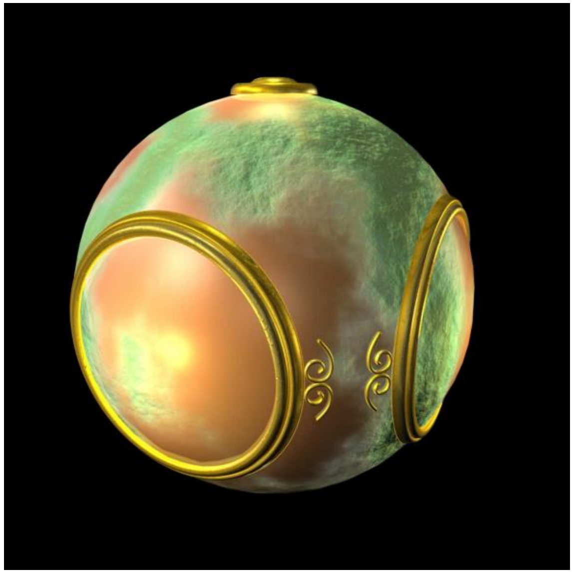
Fragment Shader Applications

Texture mapping



smooth shading

environment mapping



bump mapping

Cube Maps

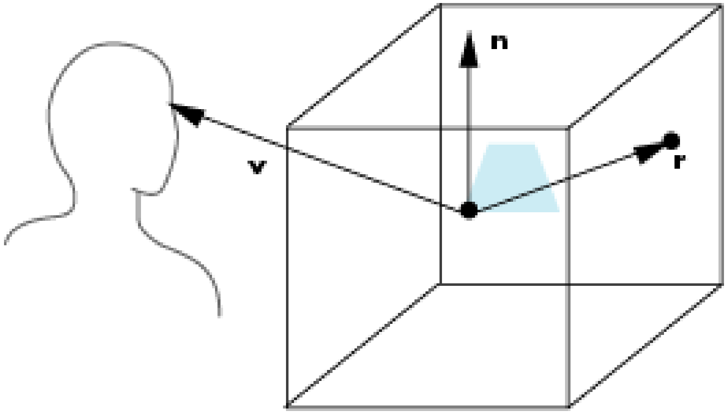
- We can form a cube map texture by defining six 2D texture maps that correspond to the sides of a box
- Supported by OpenGL
- Also supported in GLSL through cubemap sampler

```
vec4 texColor = textureCube(mycube, texcoord);
```

- Texture coordinates must be 3D

Environment Map

Use reflection vector to locate texture in cube map



Environment Maps with Shaders

- Environment map usually computed in world coordinates which can differ from object coordinates because of modeling matrix
  - May have to keep track of modeling matrix and pass it shader as a uniform variable
- Can also use reflection map or refraction map (for example to simulate water)

Reflection Map Vertex Shader

```
uniform mat4 Projection, ModelView, NormalMatrix;
in vec4 vPosition;
in vec4 normal;
out vec3 R;

void main(void)
{
  gl_Position = Projection*ModelView*vPosition;
  vec3 N = normalize(NormalMatrix*normal);
  vec4 eyePos = ModelView*gvPosition;
  R = reflect(-eyePos.xyz, N);
}
```

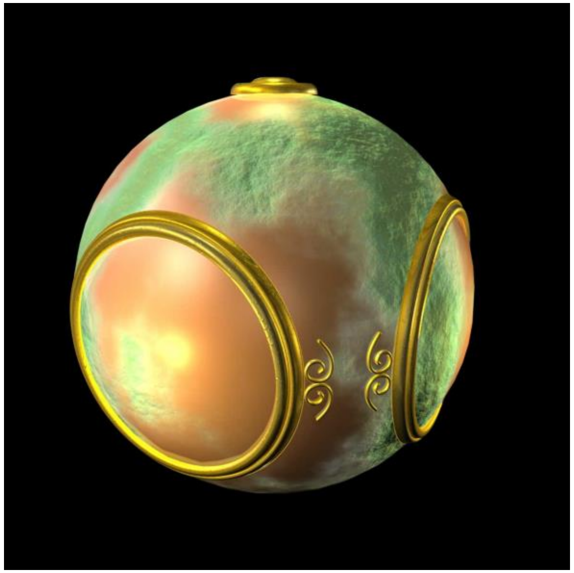
Reflection Map Fragment Shader

```
in vec3 R;
uniform samplerCube texMap;

void main(void)
{
   gl_FragColor = textureCube(texMap, R);
}
```

Bump Mapping

- Perturb normal for each fragment
- Store perturbation as textures



Normalization Maps

- Cube maps can be viewed as lookup tables 1-4 dimensional variables
- Vector from origin is pointer into table
- Example: store normalized value of vector in the map
  - Same for all points on that vector
  - Use "normalization map" instead of normalization function
  - Lookup replaces sqrt, mults and adds

Introduction

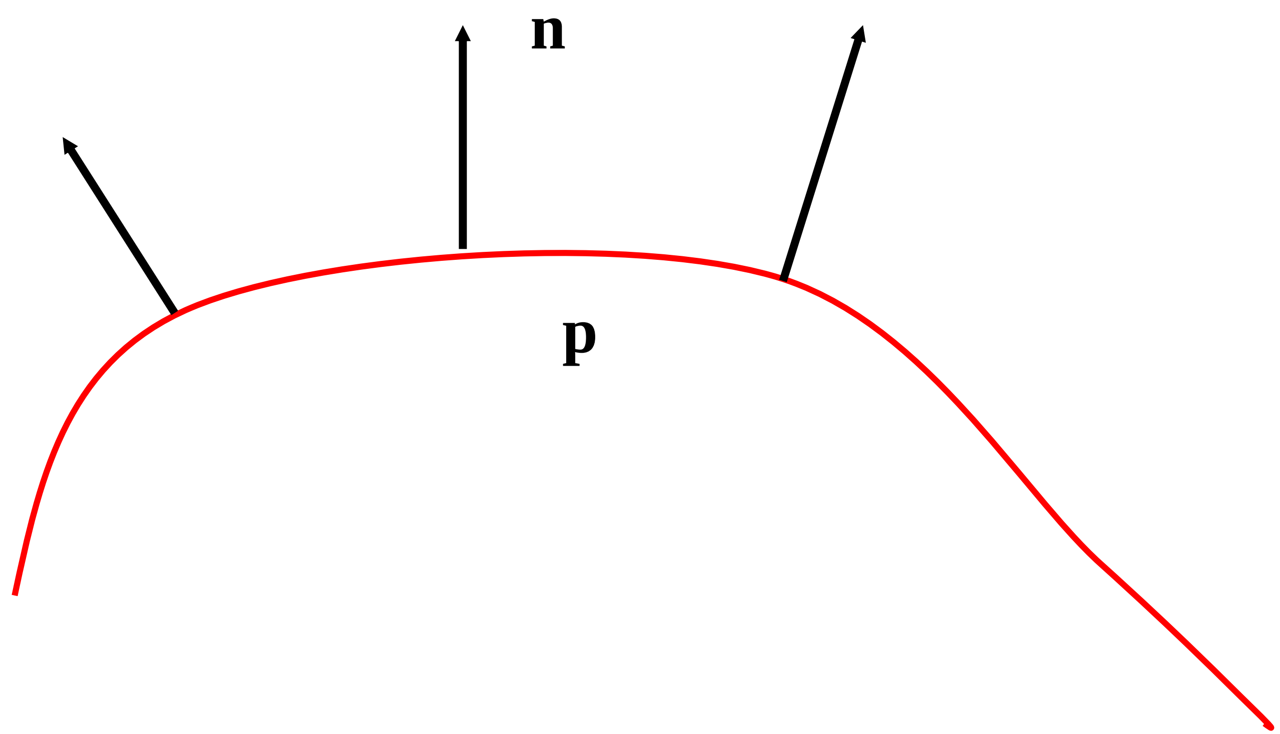
- Let's consider an example for which a fragment program might make sense
- Mapping methods
  - Texture mapping
  - Environmental (reflection) mapping
    - Variant of texture mapping
  - Bump mapping
    - Solves flatness problem of texture mapping

Modeling an Orange

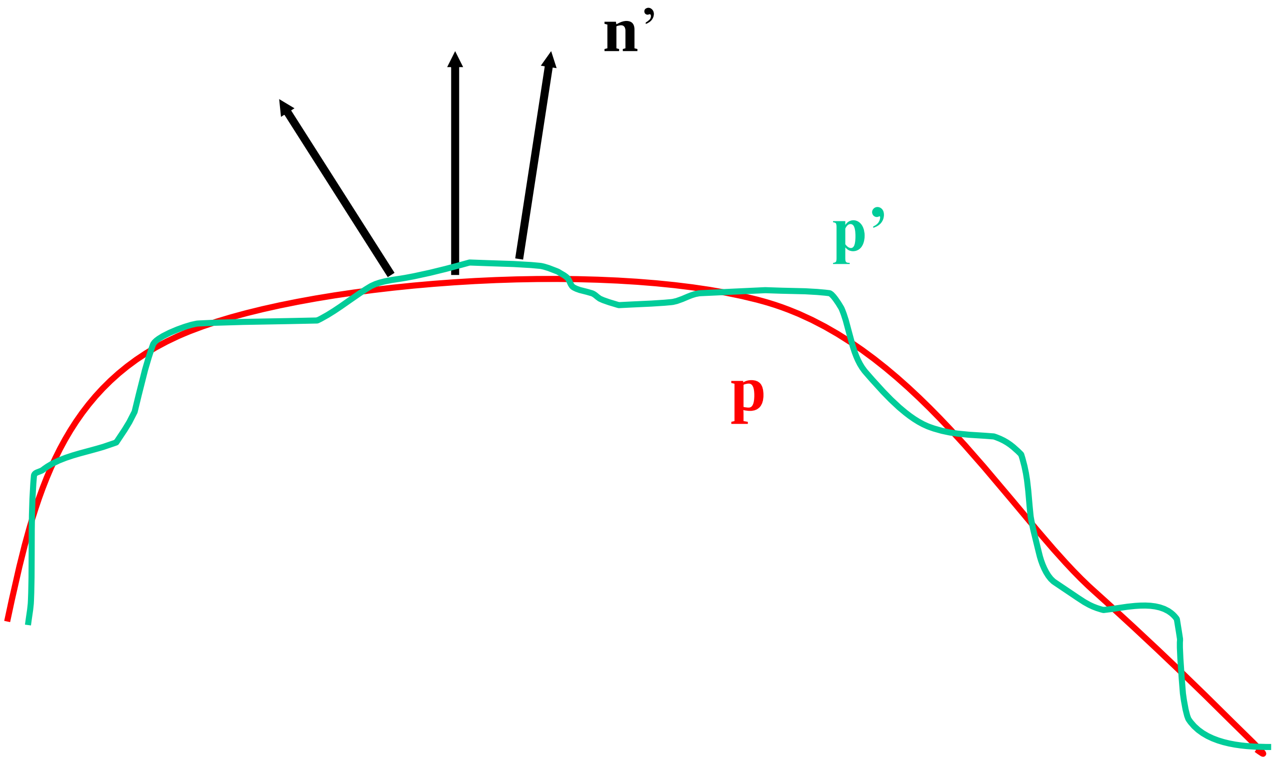
- Consider modeling an orange
- Texture map a photo of an orange onto a surface
  - Captures dimples
  - Will not be correct if we move viewer or light
  - We have shades of dimples rather than their correct orientation
- Ideally we need to perturb normal across surface of object and compute a new color at each interior point

Bump Mapping (Blinn)

- Consider a smooth surface



Rougher Version



Equations

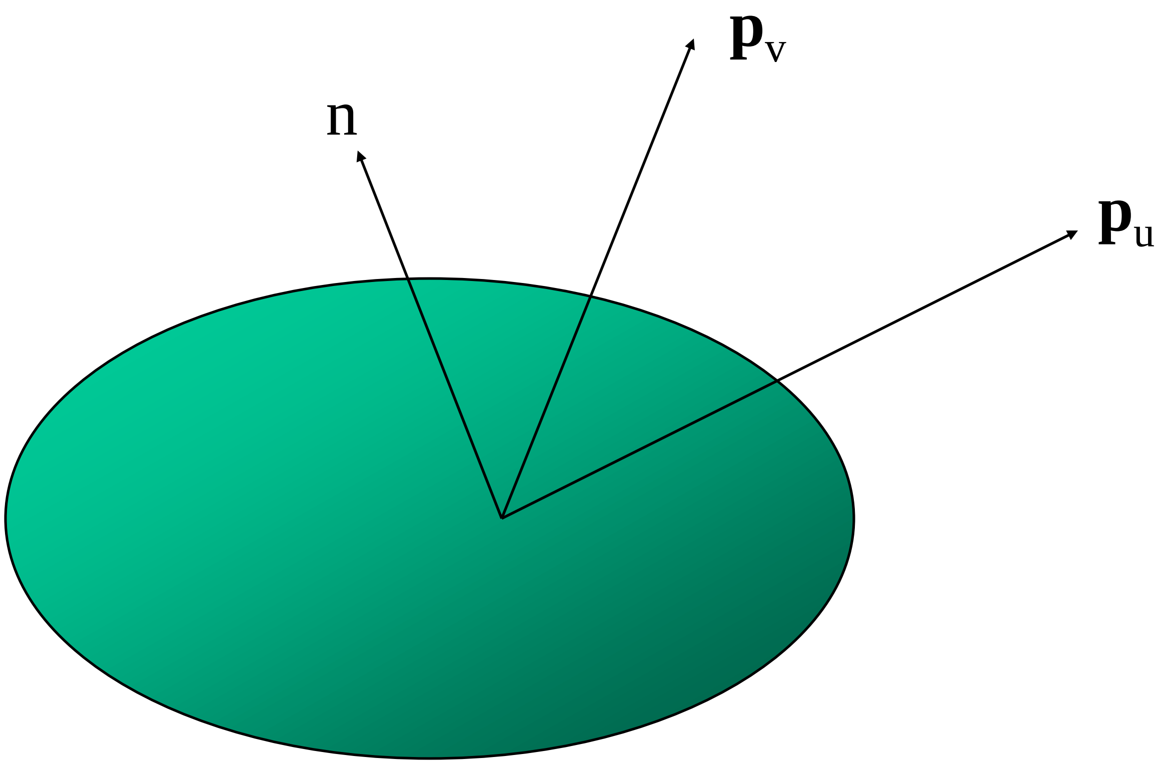
$$\mathbf{p}(u,v) = [x(u,v), y(u,v), z(u,v)]^T$$

$$\mathbf{p}_u=[\ \partial x/\ \partial u,\ \partial y/\ \partial u,\ \partial z/\ \partial u]^T$$

$$\mathbf{p}_v=[\ \partial x/\ \partial v,\ \partial y/\ \partial v,\ \partial z/\ \partial v]^T$$

$$\mathbf{n} = (\mathbf{p}_u \times \mathbf{p}_v\ )\ /\ |\ \mathbf{p}_u \times \mathbf{p}_v\ |$$

Tangent Plane



Displacement Function

$$\mathbf{p}' = \mathbf{p} + d(u,v)\,\mathbf{n}$$

d(u,v) is the bump or displacement function

$|d(u,v)| << 1$

Perturbed Normal

$$\mathbf{n'} = \mathbf{p'}_u \times \mathbf{p'}_v$$

$$\mathbf{p'}_u = \mathbf{p}_u + (\partial d/\partial u)\mathbf{n} + d(u,v)\mathbf{n}_u$$

$$\mathbf{p'}_v = \mathbf{p}_v + (\partial d/\partial v)\mathbf{n} + d(u,v)\mathbf{n}_v$$

If d is small, we can neglect last term

Approximating the Normal

$$\mathbf{n}' = \mathbf{p}'_u \times \mathbf{p}'_v$$

$$\approx \mathbf{n} + (\partial d/\partial u)\mathbf{n} \times \mathbf{p}_v + (\partial d/\partial v)\mathbf{n} \times \mathbf{p}_u$$

The vectors $\mathbf{n} \times \mathbf{p}_v$ and $\mathbf{n} \times \mathbf{p}_u$ lie in the tangent plane

Hence the normal is displaced in the tangent plane

Must precompute the arrays $\partial d/\partial u$ and $\partial d/\partial v$

Finally,we perturb the normal during shading

Image Processing

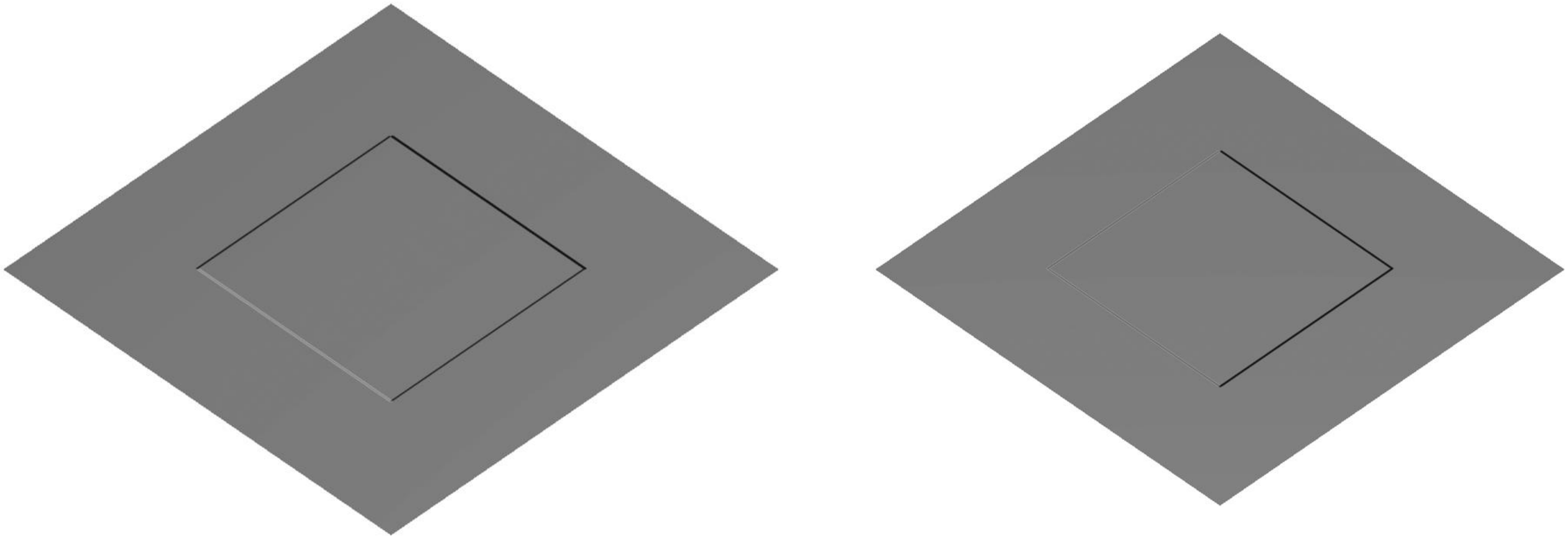
- Suppose that we start with a function d(u,v)
- We can sample it to form an array $D=[d_{ij}]$
- Then $\partial d/ \partial u \approx d_{ij} – d_{i-1,j}$

  and $\partial d/ \partial v \approx d_{ij} – d_{i,j-1}$
- **Embossing**: multipass approach using floating point buffer

The University of New Mexico

Example

Single Polygon and a Rotating Light Source



How to do this?

- The problem is that we want to apply the perturbation at all points on the surface
- Cannot solve by vertex lighting (unless polygons are very small)
- Really want to apply to every fragment
- Can’t do that in fixed function pipeline
- But can do with a fragment program!!