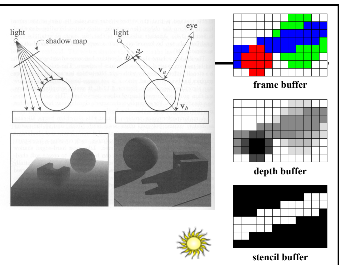
Programmable GPUS

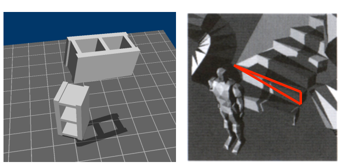
Final Project Proposals

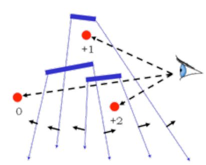
- You should all have received an email with feedback…
- Just about everyone was told:
  - Test cases weren't detailed enough
  - Project was possibly too big
  - Motivation could be strengthened
  - Use proper bibliographic citation
  - Individuals implementing refraction/rainbows should consider teaming up…
- In person/Email discussion with me and/or revised proposal suggested

Last Time?

- Planar Shadows
- Projective Texture Shadows
- Shadow Maps
- Shadow Volumes
  - Stencil Buffer





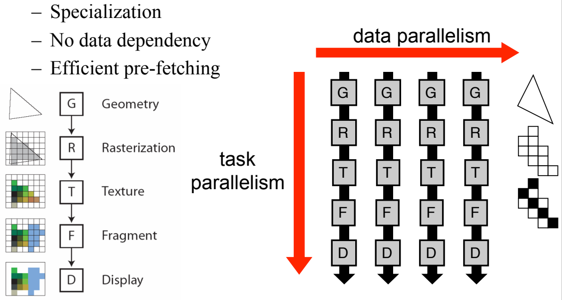


Today

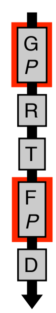
- Modern Graphics Hardware
- Shader Programming Languages
- Gouraud Shading vs. Phong Normal Interpolation
- Many "Mapping" techniques

Modern Graphics Hardware

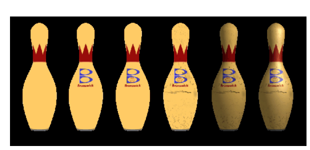
- High performance through
  - Parallelism
  - Specialization
  - No data dependency
  - Efficient pre-fetching



Programmable Graphics Hardware

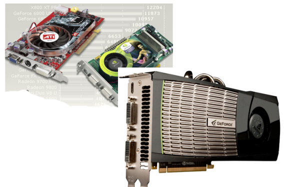


- Geometry and pixel (fragment) stage become programmable
  - Elaborate appearance
  - More and more general-purpose computation (GPU hacking)



Misc. Stats on Graphics Hardware

- 2005
  - 4-6 geometry units, 16 fragment units
  - Deep pipeline (~800 stages)
- NVIDIA GeForce 9 (Feb 2008)
  - 32/64 cores, 512 MB/1GB memory
- ATI Radeon R700 (2008)
  - 480 stream processing units
- NVIDIA GeForce GTX 480 (2010)
  - 480 cores, 1536 MB memory
  - 2560x1600 resolution
- ATI Radeon HD 7900 (2012)
  - 2048 processors, 3GB memory
- NVIDIA GeForce GTX 680 (2012)
  - 1536 cores, 2040 MB memory



Today

- Modern Graphics Hardware
- Shader Programming Languages
  - Cg design goals
  - GLSL examples
- Gouraud Shading vs. Phong Normal Interpolation
- Many "Mapping" techniques

Emerging & Evolving Languages

- Inspired by Shade Trees [Cook 1984] & Renderman Shading Language [1980's]:
  - RTSL [Stanford 2001] – real-time shading language
  - Cg [NVIDIA 2003] – "C for graphics"
  - HLSL [Microsoft 2003] – Direct X
  - GLSL [OpenGL ARB 2004] – OpenGL 2.0
  - Optix [NVIDIA 2009] – Real time ray tracing engine for CUDA
- General Purpose GPU computing
  - CUDA [NVIDIA 2007]
  - OpenCL (Open Computing Language) [Apple 2008] for heterogeneous platforms of CPUs & GPUs

Cg Design Goals

"Cg: A system for programming graphics hardware in a C-like language"
Mark et al. SIGGRAPH 2003

- Ease of programming
- Portability
- Complete support for hardware functionality
- Performance
- Minimal interference with application data
- Ease of adoption
- Extensibility for future hardware
- Support for non-shading uses of the GPU

Cg Design

- Hardware is changing rapidly [2003]… no single standard
- Specify "profile" for each hardware
  - May omit support of some language capabilities (e.g., texture lookup in vertex processor)
- Use hardware virtualization or emulation?
  - "Performance would be so poor it would be worthless for most applications"
  - Well, it might be ok for general purpose programming (not real-time graphics)

Cg compiler vs. GPU assembly

- Can inspect the assembly language produced by Cg compiler and perform additional optimizations by hand
  - Generally once development is complete (& output is correct)
- Using Cg is easier than writing GPU assembly from scratch

(Typical) Language Design Issues

- Parameter binding
- Call by reference vs. call by value
- Data types: 32 bit float, 16 bit float, 12 bit fixed & type-promotion (aim for performance)
- Specialized arrays or general-purpose arrays
  - `float4 x` vs. `float x[4]`
- Indirect addressing/pointers (not allowed…)
- Recursion (not allowed…)

Today

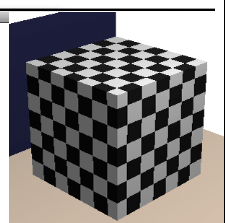
- Modern Graphics Hardware
- Shader Programming Languages
  - Cg design goals
  - GLSL examples
- Gouraud Shading vs. Phong Normal Interpolation
- Many “Mapping” techniques

GLSL example: checkerboard.vs (GLUT)

Emacs@tony.dyn.cs.rpi.edu

```
varying vec3 normal;
varying vec3 position_eyespace;
varying vec3 position_worldspace;

// a shader for a black & white checkerboard

void main(void) {
  position_eyespace = vec3(gl_ModelViewMatrix * gl_Vertex);
  position_worldspace = gl_Vertex.xyz;
  normal = normalize(gl_NormalMatrix * gl_Normal);
  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

--:--- checkerboard.vs All L13 CVS-1.1 (C/l Abbrev)

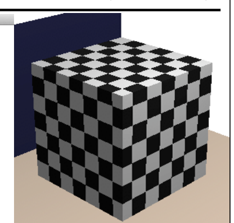


GLSL example: hw4_shader.vs (GLFW)

hw4_shader.vs

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec3 vertexNormal_modelspace;
layout(location = 2) in vec3 vertexColor;

// Output data
out vec3 vertexPosition_worldspace;
out vec3 vertexNormal_worldspace;
out vec3 vertexNormal_cameraspace;
out vec3 EyeDirection_cameraspace;
out vec3 myColor;

// Values that stay constant for the whole mesh.
uniform mat4 MVP;
uniform mat4 V;
uniform mat4 M;
uniform vec3 LightPosition_worldspace;

void main(){

  // Output position of the vertex, in clip space : MVP * position
  gl_Position =  MVP * vec4(vertexPosition_modelspace,1);

  // Position of the vertex, in worldspace : M * position
  vertexPosition_worldspace = (M * vec4(vertexPosition_modelspace,1)).xyz;

  // Vector that goes from the vertex to the camera, in camera space.
  // In camera space, the camera is at the origin (0,0,0).
  vec3 vertexPosition_cameraspace = ( V * M * vec4(vertexPosition_modelspace,1)).xyz;

  EyeDirection_cameraspace = vec3(0,0,0) - vertexPosition_cameraspace;

  vertexNormal_worldspace = normalize (M * vec4(vertexNormal_modelspace,0)).xyz;

  // pass color to the fragment shader
  myColor = vertexColor;
}
```

-:--- hw4_shader.vs All L1 Git-master (C++/l Abbrev)



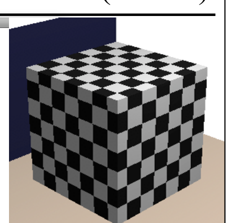
GLSL example: checkerboard.fs (GLUT)

Emacs@tony.dyn.cs.rpi.edu

```
varying vec3 normal;
varying vec3 position_eyespace;
varying vec3 position_worldspace;

// a shader for a black & white checkerboard

void main (void) {

  vec3 color;

  // determine the parity of this point in the 3D checkerboard
  int count = 0;
  if (mod(position_worldspace.x,0.3)> 0.15) count++;
  if (mod(position_worldspace.y,0.3)> 0.15) count++;
  if (mod(position_worldspace.z,0.3)> 0.15) count++;
  if (count == 1 || count == 3) {
    color = vec3(0.1,0.1,0.1);
  } else {
    color = vec3(1,1,1);
  }

  // direction to the light
  vec3 light = normalize(gl_LightSource[1].position.xyz - position_eyespace);

  // basic diffuse
  float ambient = 0.3;
  float diffuse = 0.7*max(dot(normal,light),0.0);
  color = ambient*color + diffuse*color;
  gl_FragColor = vec4 (color, 1.0);
}
```

--:--- checkerboard.fs All L1 CVS-1.1 (C/l Abbrev)

ESC <mouse-1> is undefined

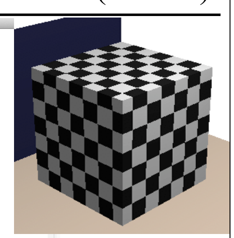


GLSL example: hw4_shader.fs (GLFW)

hw4_shader_checkerboard.fs

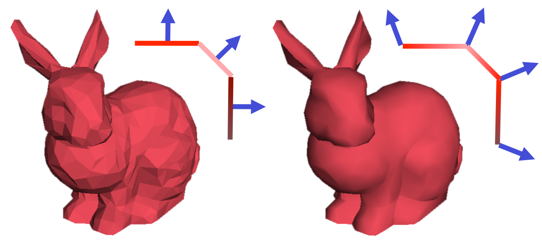
```
in vec3 vertexNormal_worldspace;

// Ouput data
out vec3 color;

// Values that stay constant for the whole mesh.
uniform vec3 LightPosition_worldspace;
uniform int colormode;
uniform int whichshader;

// ----------------------------------------------
// a shader for a black & white checkerboard
vec3 checkerboard(vec3 pos) {
  // determine the parity of this point in the 3D checkerboard
  int count = 0;
  if (mod(pos.x,0.3)> 0.15) count++;
  if (mod(pos.y,0.3)> 0.15) count++;
  if (mod(pos.z,0.3)> 0.15) count++;
  if (count == 1 || count == 3) {
    return vec3(0.1,0.1,0.1);
  } else {
    return vec3(1,1,1);
  }
}

// ----------------------------------------------
void main(){

  vec3 LightColor = vec3(1,1,1);
  float LightPower = 4.0f;

  // surface normal
  vec3 surface_normal =  vertexNormal_worldspace;

  // Material properties
  vec3 MaterialDiffuseColor = myColor;
  if (whichshader == 1) {
    MaterialDiffuseColor = checkerboard(vertexPosition_worldspace);
  } else if (whichshader == 2) {
    vec3 normal2;
    MaterialDiffuseColor = orange(vertexPosition_worldspace,surface_normal);
  } else if (whichshader == 3) {
    MaterialDiffuseColor = wood(vertexPosition_worldspace,surface_normal);
  }
```

-:--- hw4_shader_checkerboard.fs 5% L7 (C++/l Abbrev)



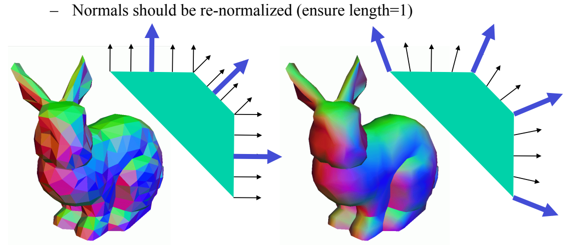
Remember Gouraud Shading?

- Instead of shading with the normal of the triangle, we'll shade the vertices with the *average normal* and *interpolate the shaded color* across each face
  - This gives the *illusion of a smooth surface* with smoothly varying normals



Phong Normal Interpolation (Not Phong *Shading*)

- *Interpolate the average vertex normals* across the face and compute *per-pixel shading*
  - Normals should be re-normalized (ensure length=1)



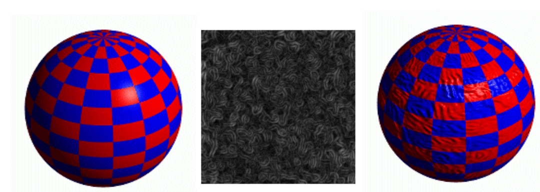
- Before shaders, per-pixel shading was not possible in hardware (Gouraud shading is actually a decent substitute!)

Today

- Modern Graphics Hardware
- Shader Programming Languages
- Gouraud Shading vs. Phong Normal Interpolation
- Many "Mapping" techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping

Bump Mapping

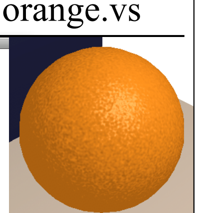
- Use textures to alter the surface normal
  - Does not change the actual shape of the surface
  - Just shaded as if it were a different shape



Sphere w/Diffuse Texture — Swirly Bump Map — Sphere w/Diffuse Texture & Bump Map

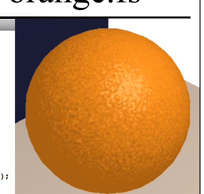
Another GLSL example: orange.vs

```
varying vec3 normal;
varying vec3 position_eyespace;
varying vec3 position_worldspace;

// a shader that looks like orange peel

void main(void) {

  // the fragment shader requires both the world space position (for
  // consistent bump mapping) & eyespace position (for the phong
  // specular highlight)
  position_eyespace = vec3(gl_ModelViewMatrix * gl_Vertex);
  position_worldspace = gl_Vertex.xyz;

  // pass along the normal
  normal = normalize(gl_NormalMatrix * gl_Normal);

  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
 }
```



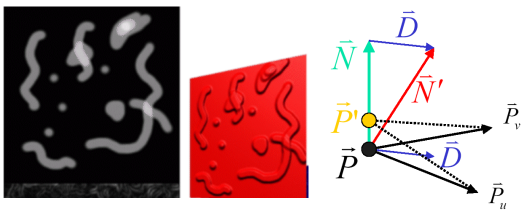
Another GLSL example: orange.fs

```
varying vec3 normal;
varying vec3 position_eyespace;
varying vec3 position_worldspace;

// a shader that looks like orange peel

void main (void) {

  // the base color is orange!
  vec3 color = vec3(1.0,0.5,0.1);

  // high frequency noise added to the normal for the bump map
  vec3 normal2 = normalize(normal+0.4*noise3(70.0*position_worldspace));

  // direction to the light
  vec3 light = normalize(gl_LightSource[1].position.xyz - position_eyespace);
  // direction to the viewer
  vec3 eye_vector = normalize(-position_eyespace);
  // ideal specular reflection
  vec3 reflected_vector = normalize(-reflect(light,normal2));

  // basic phong lighting
  float ambient = 0.6;
  float diffuse = 0.4*max(dot(normal2,light),0.0);
  float specular = 0.2 * pow(max(dot(reflected_vector,eye_vector),0.0),10.0);
  vec3 white = vec3(1.0,1.0,1.0);
  color = ambient*color + diffuse*color + specular*white;
  gl_FragColor = vec4 (color, 1.0);
}
```

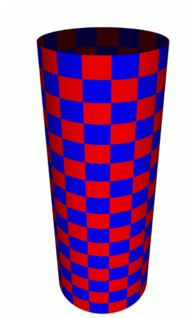


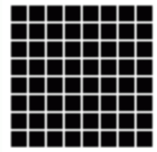
Bump Mapping

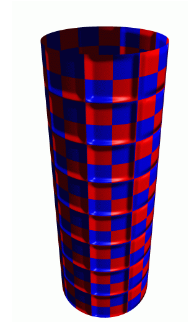
- Treat a greyscale texture as a single-valued height function
- Compute the normal from the partial derivatives in the texture



Another Bump Map Example

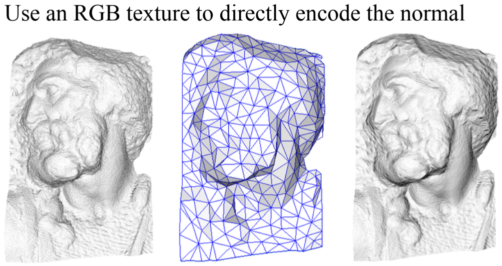

Cylinder w/Diffuse Texture Map


Bump Map


Cylinder w/Texture Map & Bump Map

Normal Mapping

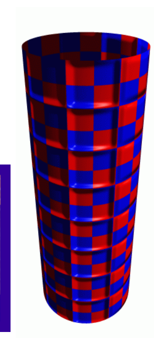
- Variation on Bump Mapping:
  Use an RGB texture to directly encode the normal



http://en.wikipedia.org/wiki/File:Normal_map_example.png

What's Missing?

- There are no bumps on the silhouette of a bump-mapped or normal-mapped object
- Bump/Normal maps don't allow self-occlusion or self-shadowing

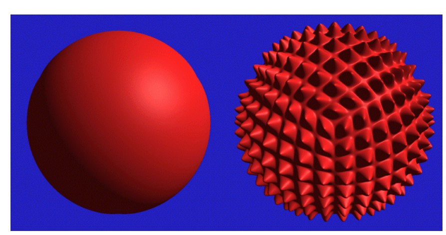


Today

- Modern Graphics Hardware
- Shader Programming Languages
- Gouraud Shading vs. Phong Normal Interpolation
- Many "Mapping" techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping

Displacement Mapping

- Use the texture map to actually move the surface point
- The geometry must be displaced before visibility is determined



Displacement Mapping

Image from:

Geometry Caching for Ray-Tracing Displacement Maps
EGRW 1996
Matt Pharr and Pat Hanrahan

note the detailed shadows cast by the stones

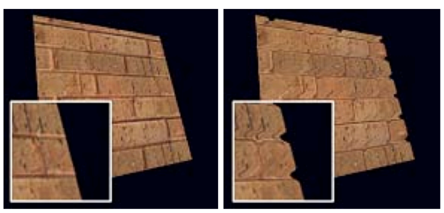
Displacement Mapping

Ken Musgrave

Parallax Mapping

a.k.a. Offset Mapping or Virtual Displacement Mapping

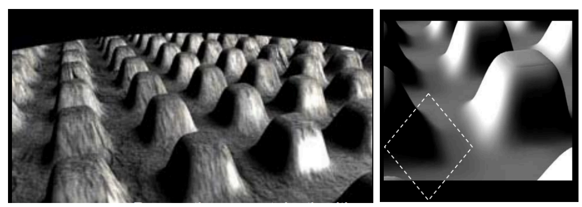
- Displace the texture coordinates for each pixel based on view angle and value of the height map at that point
- At steeper view-angles, texture coordinates are displaced more, giving illusion of depth due to parallax effects



"Detailed shape representation with parallax mapping", Kaneko et al. ICAT 2001

Parallax Occlusion Mapping

- Brawley & Tatarchuk 2004
- Per pixel ray tracing of the heightfield geometry
- Occlusions & soft shadows



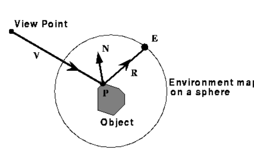
http://developer.amd.com/media/gpu_assets/Tatarchuk-ParallaxOcclusionMapping-Sketch-print.pdf

Today

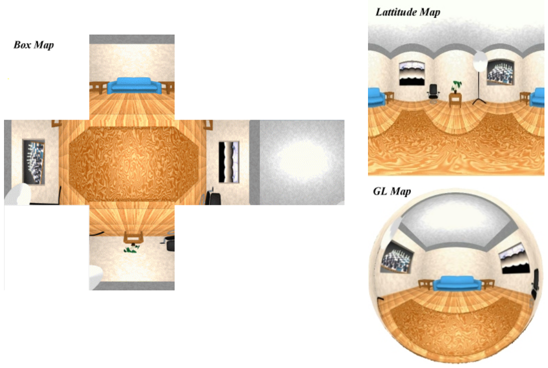
- Modern Graphics Hardware
- Shader Programming Languages
- Gouraud Shading vs. Phong Normal Interpolation
- Many "Mapping" techniques
  - Bump Mapping
  - Displacement Mapping
  - Environment Mapping
  - Light Mapping
  - Normal Mapping
  - Parallax Mapping
  - Parallax Occlusion Mapping

Environment Maps

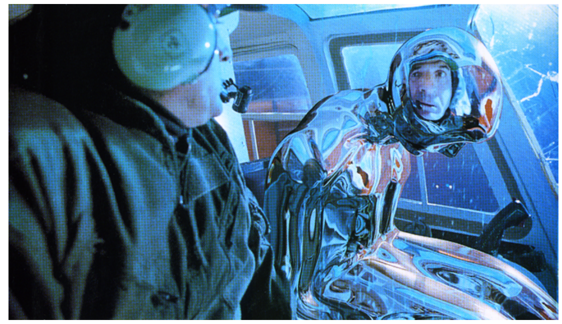
- We can simulate reflections by using the direction of the reflected ray to index a spherical texture map at "infinity".
- Assumes that all reflected rays begin from the same point.



What's the Best Chart?



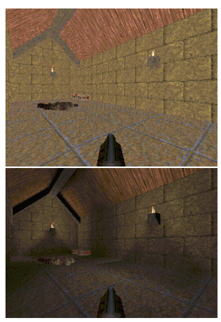
Environment Mapping Example



Terminator II

Texture Maps for Illumination

- Also called "Light Maps"



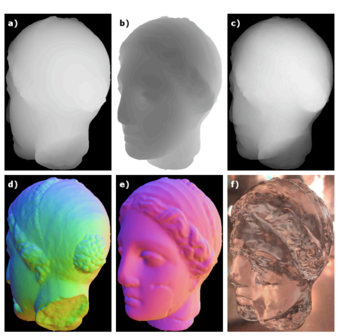
Quake

Questions?

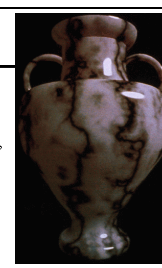
Image by Henrik Wann Jensen
Environment map by Paul Debevec

Reading for Today:

- Chris Wyman, "An Approximate Image-Space Approach for Interactive Refraction", SIGGRAPH 2005



Readings for Friday:



Choose:

- “An Image Synthesizer”, Perlin, SIGGRAPH 1985 & “Improving Noise”, Perlin, SIGGRAPH 2002
- “Procedural Modeling of Buildings” Mueller, Wonka, Haegler, Ulmer & Van Gool, SIGGRAPH 2006