



Per pixel lighting Parallel programming



Today's lectures

- Per pixel lighting
 - Learning by examples
 - Ambient/diffuse shading
 - Bump mapping
 - Compiling and debugging
- Parallel programming
 - General concepts (pmn)
 - The GPU as a parallel processor
- Weekly assignment

Changing tuesday's schedule?

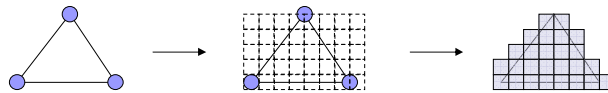
- We hold on to meeting from 14-15.
- However, we CANCEL presentation at 2/11
 - We will have two weekly assignments presented and discussed at 9/11 (groups 1-2/4-5).

Assigning groups

■ 19970561 Thomas Rasmussen -	
19984373 Niels Jeppe Anders Nielsen	1
19990742 Erik Søren Sørensen -	
20001263 Rasmus Witting Larsen	2
20001467 Bent Bisballe	3
20001683 Søren Gjellerup Christiansen	3
20001995 Karsten stergaard Noe	4
20002177 Peter Gade Jensen	2
20002482 Niels Carsten Thrane	2
20002751 Lars Ole Simonsen	2
20003750 Mads Østerby -	
20022647 Michael Bræmer Nielsen	1
20022872 Kasper Langer -	
20023690 Jonas Larsen -	
20012938 Peter Trier	4
20011373 Thomas Mølhav	4
19982078 Nicolai Lundgaard	1
20023502 Søren Skov	1
19960197 Jesper Fruegaard Andersen	3
19970541 Aske Simon Christensen	5
19991875 Michael Westergaard	5
19971799 Allan Rasmusson	5
19990498 Troels Bjerre Sørensen	5
19992541 Daniel Nielsen	3

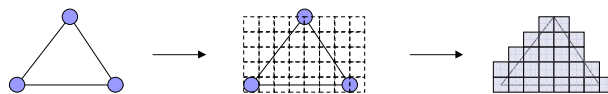
The sample application from tuesday

- Fixed OpenGL pipeline
 - Gouraud or flat shading
 - Lighting calculated by the vertex program
 - Vertex colors interpolated by the rasterizer
 - Modulated with texture in fragment program



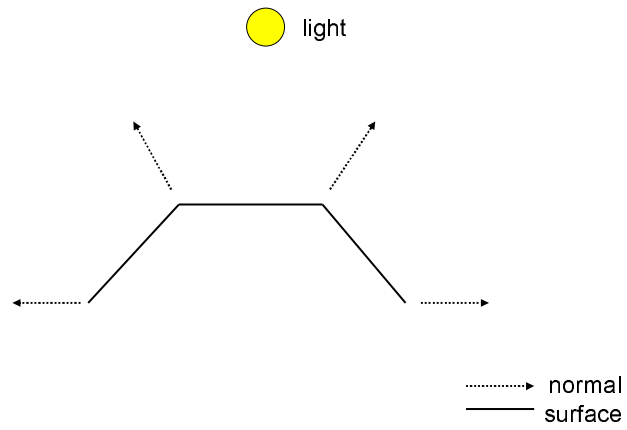
Per pixel lighting

- For each fragment, how do we find the
 - Position
 - Normal
 - Light/material parameters
- and compute lighting on a per-fragment basis?

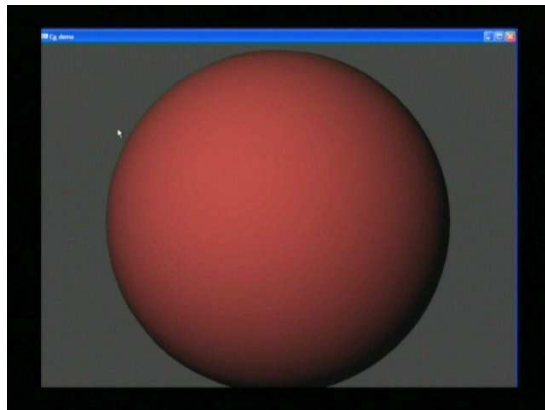


Phong shading

An alternative to denser triangulation



Per pixel normals



Per pixel normals - structs

```
// Define interface between the application and the vertex program
struct app_vertex {
    float4 Position : POSITION;
    float4 Normal   : NORMAL;
    float4 Color    : COLOR0;
    float4 TexCoord : TEXCOORD0;
};

// Define the interface between the vertex- and the fragment programs
struct vertex_fragment {
    float4 Position : POSITION; // For the rasterizer
    float4 NormalE  : TEXCOORD0;
    float4 PositionE : TEXCOORD1;
};

// The final result written to the framebuffer
struct fragment_out {
    float4 Color : COLOR0;
};
```

Per pixel normals – vertex program

```
vertex_fragment
vertex_main( app_vertex IN ){

    vertex_fragment OUT;

    // Get OpenGL state matrices
    float4x4 ModelView = glstate.matrix.modelview[0];
    float4x4 ModelViewIT = glstate.matrix.invtans.modelview[0];
    float4x4 ModelViewProj = glstate.matrix.mvp;

    // Transform vertex
    OUT.Position = mul( ModelViewProj, IN.Position );

    // Pass on the vertex normal as a texture coordinate
    OUT.NormalE.xyz = normalize( mul( (float3x3)ModelViewIT, IN.Normal.xyz ) );
    OUT.PositionE = mul( ModelView, IN.Position );

    return OUT;
}
```

Per pixels normals – fragment program

```
fragment_out
fragment_main( vertex_fragment IN, uniform sampler2D Texture2D,
    uniform float4 lightAmbient, uniform float4 lightDiffuse,
    uniform float4 materialAmbient, uniform float4 materialDiffuse, uniform float4 lightPosition )
{
    fragment_out OUT;

    // Use only ambient/diffuse shading. Calculate in eye-space.
    float3 n = normalize( IN.NormalE.xyz );
    float4 position = IN.PositionE;

    // float3 L = normalize( glstate.light[0].position - position ).xyz;
    float3 L = normalize( lightPosition - position ).xyz;
    float diffuseLight = max( dot( L, n ), 0 );

    // OUT.Color = glstate.light[0].ambient * glstate.material.ambient +
    //             glstate.light[0].diffuse * glstate.material.diffuse * diffuseLight;

    OUT.Color = lightAmbient * materialAmbient +
        lightDiffuse * materialDiffuse * diffuseLight;

    return OUT;
}
```

Per pixel normals - runtime

```
static void DrawGeometry()
{
    float Ka[4] = { .05, .05, .05, 1 };
    float Kd[4] = { .7, .2, .2, 1 };
    float lightPos[4] = { -2, 2, 0, 1 };
    float lightColor[4] = { 1, 1, 1, 1 };

    // Enable and set input parameters
    cgGLEnableTextureParameter(cgGetNamedParameter(fragmentProgram, "Texture2D"));
    cgGLSetParameter4fv( cgGetNamedParameter( fragmentProgram, "lightPosition"), lightPos );
    cgGLSetParameter4fv( cgGetNamedParameter( fragmentProgram, "lightAmbient"), lightColor );
    cgGLSetParameter4fv( cgGetNamedParameter( fragmentProgram, "lightDiffuse"), lightColor );
    cgGLSetParameter4fv( cgGetNamedParameter( fragmentProgram, "materialAmbient"), Ka );
    cgGLSetParameter4fv( cgGetNamedParameter( fragmentProgram, "materialDiffuse"), Kd );

    // And draw the geometry.
    glutSolidSphere ( 2, 16, 16 );

    // Be a good citizen and disable the texture binding we set up above.
    cgGLDisableTextureParameter(param);
}
```

Bump mapping

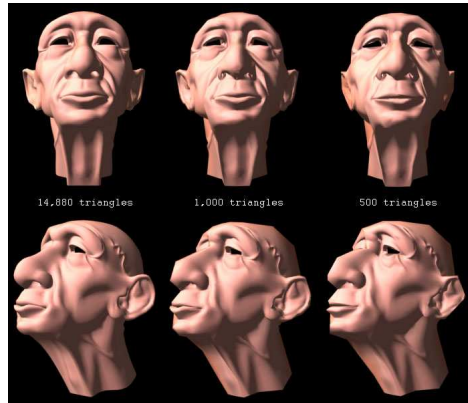
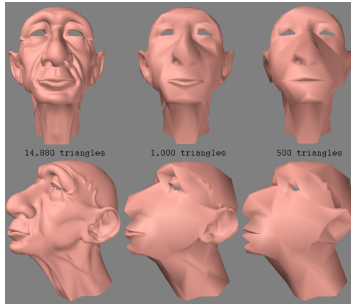
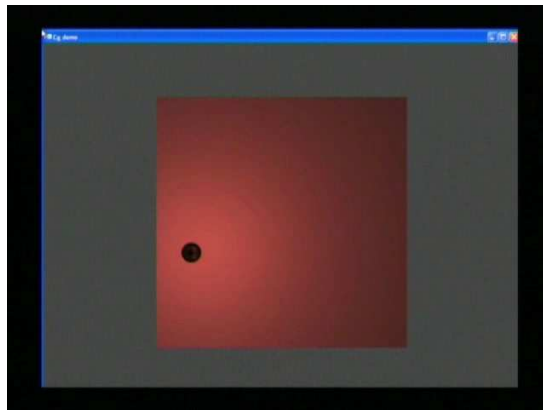


Image source:
<http://amber.rc.arizona.edu/lw/normalmaps.html>

Bump mapping sample application



Per pixels normals – fragment program

```
fragment_out
fragment_main( vertex_fragment IN,
               uniform sampler2D Texture2D,
               uniform float4 lightAmbient, uniform float4 lightDiffuse,
               uniform float4 materialAmbient, uniform float4 materialDiffuse,
               uniform float4 lightPosition )
{
    fragment_out OUT;

    // Use only ambient/diffuse shading. Calculate in eye-space.
    float3 n = normalize( tex2D( Texture2D, IN.TexCoord.xy ).xyz * 2.0 - 1.0f );
    float4 position = IN.PositionE;
    float3 L = normalize( lightPosition - position ).xyz;
    float diffuseLight = max( dot( L, n ), 0 );

    OUT.Color = lightAmbient * materialAmbient +
               lightDiffuse * materialDiffuse * diffuseLight;

    return OUT;
}
```

Compiling Cg programs

`cgc -profile arbvp1 -entry vertex_main simple_bumpmap.cg`

```
simple_bumpmap.cg
66 lines, 0 errors.
IARBvp1.0
# ARB_vertex_program generated by NVIDIA Cg compiler
# cgc Version 1.2.1001, build date Mar 17 2004 10:32:28
# command line args: -profile arbvp1 -entry vertex_main
# nv3Dvp backend compiling 'vertex_main' program
# vendor NVIDIA Corporation
# version 1.0.02
# profile arbvp1
# program vertex_main
# var float4 IN.Position : $vin.POSITION : POSITION : 0 : 1
# var float4 IN.TexCoord : $vin.TEXCOORD0 : TEXCOORD0 : 0 : 1
# var float4 vertex_main.Position : $vout.POSITION : HPOS : -1 : 1
# var float4 vertex_main.TexCoord : $vout.TEXCOORD0 : TEXD : -1 : 1
# var float4 vertex_main.PositionE : $vout.TEXCOORD1 : TEX1 : -1 : 1

ATTRIB v24 = vertex.texcoord[0];
ATTRIB v16 = vertex.position;
PARAM s259[4] = { state.matrix.mvp };
PARAM s223[4] = { state.matrix.modelview[0] };
    MOV result.texcoord[0], v24;
    DP4 result.position.x, s259[0], v16;
    DP4 result.position.y, s259[1], v16;
    DP4 result.position.z, s259[2], v16;
    DP4 result.position.w, s259[3], v16;
    DP4 result.texcoord[1].x, s223[0], v16;
    DP4 result.texcoord[1].y, s223[1], v16;
    DP4 result.texcoord[1].z, s223[2], v16;
    DP4 result.texcoord[1].w, s223[3], v16;

END
# 9 instructions
# 0 temp registers
# End of program
```

Compiling Cg programs

```
cgc -profile arbfp1 -entry fragment_main simple_bumpmap.cg
```

```
66 lines, 0 errors.
!!ARBfp1.0
#...
PARAM u0 = program.local[0];
PARAM u2 = program.local[2];
PARAM u1 = program.local[1];
PARAM u3 = program.local[3];
PARAM u4 = program.local[4];
PARAM c0 = {0, 2, 1, 0};
TEMP R0;
TEMP R1;

TEX R0.xyz, fragment.texcoord[0], texture[0], 2D;
MAD R0.xyz, R0, c0.y, -c0.z;
ADD R1, u4, -fragment.texcoord[1];
DP4 R0.w, R1, R1;
RSQ R0.w, R0.w;
MUL R1.xyz, R0.w, R1;
DP3 R0.w, R0, R0;
RSQ R0.w, R0.w;
MUL R0.xyz, R0.w, R0;
DP3 R0.x, R1, R0;
MAX R0.x, R0.x, c0.x;
MOV R1, u3;
MUL R1, u2, R1;
MUL R0, R1, R0.x;
MOV R1, u1;
MAD result.color, u0, R1, R0;
END
# 16 instructions, 2 R-reg, 0 H-reg.
# End of program
```

Debugging Cg programs

- Difficult!
 - No printf available
- Use `glReadPixels()` to read back the rendering buffer to the CPU - then print it
- Use `OUT.Color` in fragment programs to debug float vectors visually
 - `n`, `L`, `diffuseLight` etc.
 - Use `abs()` to see negative values
 - Clamp to `[0;1]`