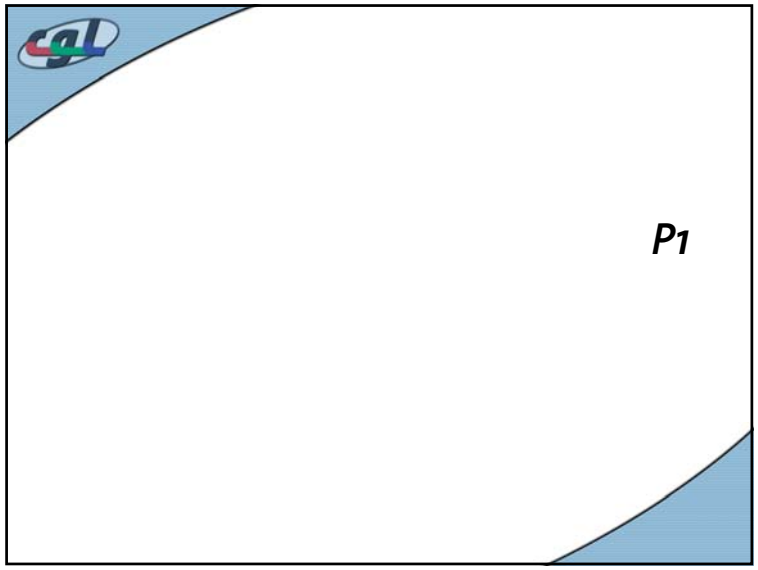
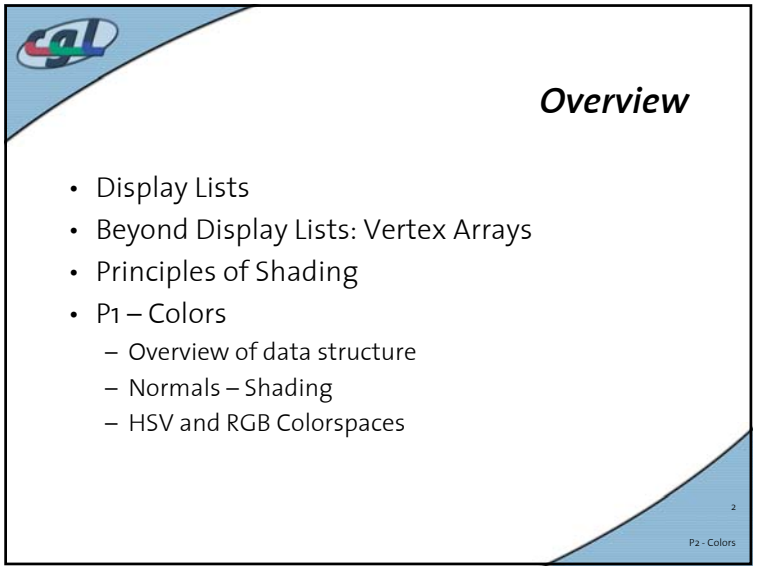
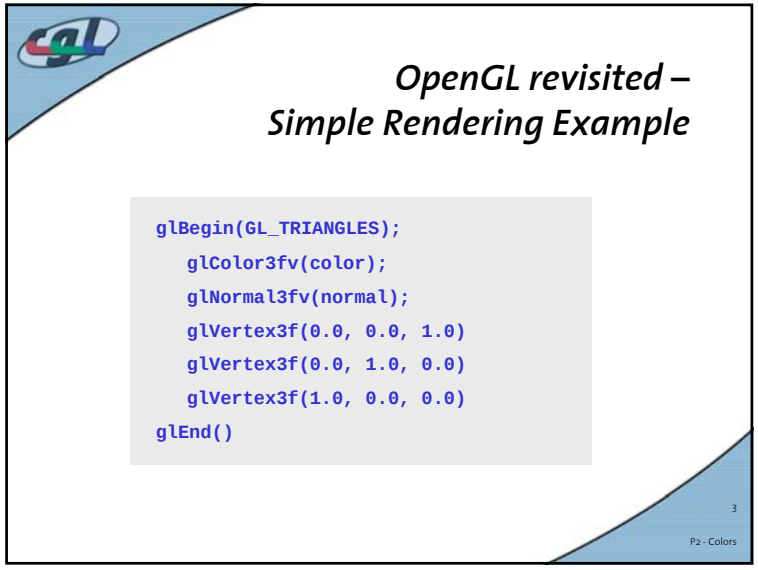


P1



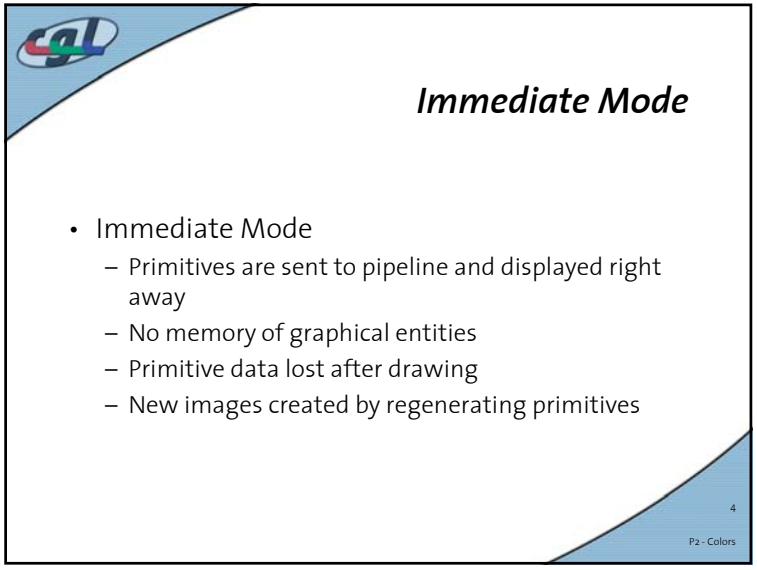
Overview

- Display Lists
- Beyond Display Lists: Vertex Arrays
- Principles of Shading
- P1 – Colors
  - Overview of data structure
  - Normals – Shading
  - HSV and RGB Colorspaces



OpenGL revisited – Simple Rendering Example

```
glBegin(GL_TRIANGLES);
   glColor3fv(color);
   glNormal3fv(normal);
   glVertex3f(0.0, 0.0, 1.0)
   glVertex3f(0.0, 1.0, 0.0)
   glVertex3f(1.0, 0.0, 0.0)
glEnd()
```

Immediate Mode

- Immediate Mode
  - Primitives are sent to pipeline and displayed right away
  - No memory of graphical entities
  - Primitive data lost after drawing
  - New images created by regenerating primitives

Retained Mode

- Retained Mode Graphics
  - Primitives stored in display lists (in "compiled" form)
  - Display lists kept on graphics server
  - Images recreated by "executing" the display list
  - More efficient
  - Can be redisplayed with different state
  - Can be shared among OpenGL graphics contexts

Display Lists I

- **steps:** ***create it, then call it***

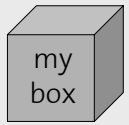
```
GLuint id;
void init( void )
{
   id = glGenLists( 1 );
   glNewList( id, GL_COMPILE );
   // other OpenGL routines
   glEndList();
}
void display( void )
{
   glCallList( id );
}
```


Display Lists II – glNewList()

- `glNewList` accepts the constants:
  - `GL_COMPILE`, which creates a display list
  - `GL_COMPILE_AND_EXECUTE`, which both creates and executes a display list
- Lists can be overwritten
  - Creation of a new list with the same identifying number as an existing display list
  - No error occurs.

Display Lists III – Modeling

```
glNewList( BOX, GL_COMPILE )
   glBegin( GL_QUADS );
      glVertex3d( … );
      glVertex3d( … );
      …
   glEnd();
glEndList();
```



Display Lists IV – Advanced

- Not all OpenGL routines can be stored in display lists
- State changes persist, even after a display list is finished
- Display lists can call other display lists
- Display lists are not editable, but you can fake it
  - make a list (A) which calls other lists (B, C, and D)
  - delete and replace B, C, and D, as needed

A Step Beyond – Vertex Arrays

- Edges and Vertices are often processed multiple times
  - Box: 6 faces, 8 vertices, each vertex is processed 3 times, thus **24 points** processed!
- OpenGL has vertex array functions
  - Enables collective access to data
  - Minimization of function calls
  - Dramatic Performance gain

Vertex Arrays – How to use them

- 3 steps:
  - Activate up to 6 vertex arrays
    - Coordinate, Color, Normal, etc.
  - Fill the arrays with data
  - Render geometry with array data
- Vertex data can be **dynamically changed**
- OpenGL 1.1+ needed

Vertex Arrays – Activation

- `glEnableClientState(GLenum array)`
  - Activates certain array
  - Allowed arrays:
    - `GL_VERTEX_ARRAY`
    - `GL_COLOR_ARRAY`
    - `GL_NORMAL_ARRAY`
    - `GL_INDEX_ARRAY`
    - `GL_TEXTURE_COORD_ARRAY`
    - `GL_EDGE_FLAG_ARRAY`
- `glDisableClientState(GLenum array)`
  - Deactivates certain array

Vertex Arrays – Data

- 6 different functions for specifying array data

```
glVertexPointer(GLint size, GLenum type,
    GLsizei stride, const GLvoid * pointer)
```

  - size: number of channels (2, 3 or 4)
  - type: data type (**GL_SHORT**, **GL_INT**, **GL_FLOAT**, etc.)
  - stride: byte offset between consecutive data
  - pointer: pointer to first data item
- More…

```
glColorPointer(…)
glNormalPointer(…)
```


Vertex Arrays – Rendering

- Single array element

```
glArrayElement(GLint ith)
```

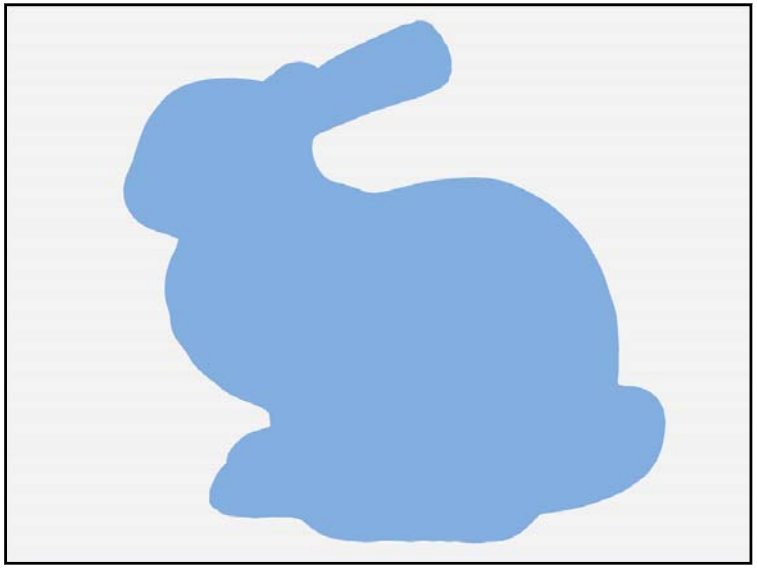
  - Gets values from every activated array
  - Renders element
  - Stands in a `glBegin()` – `glEnd()` pair
- More Elements

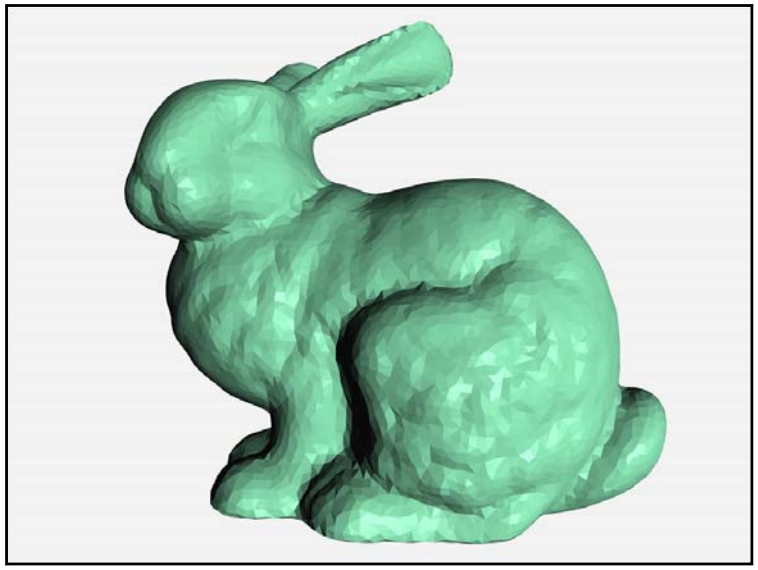
```
glDrawElements(GLenum mode,
  GLsizei count, GLenum type, void *indices)
```

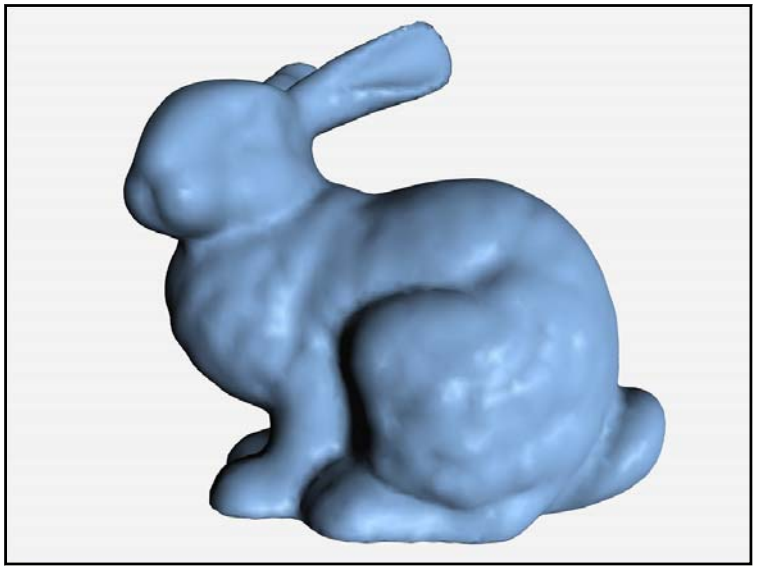
  - Access to data elements over indices array
  - Other possibility:

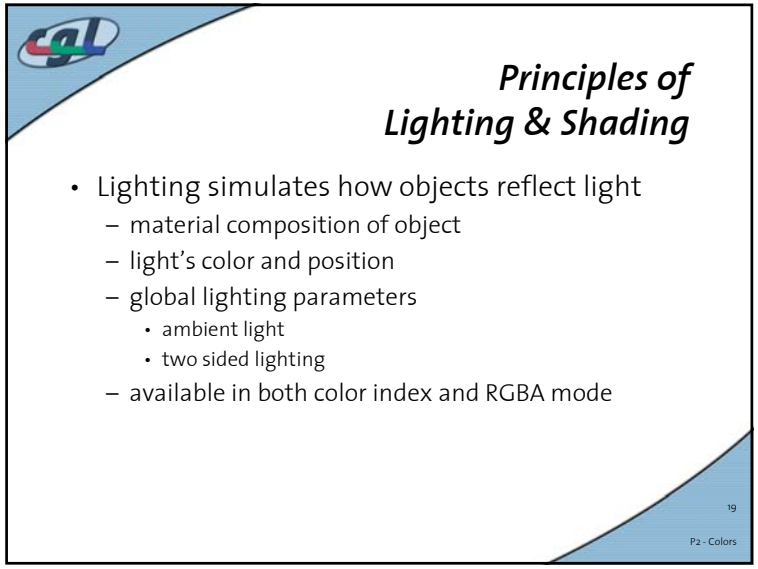
```
glDrawArrays(GLenum mode, GLint first,
  GLsizei count)
```


Principles of Lighting & Shading



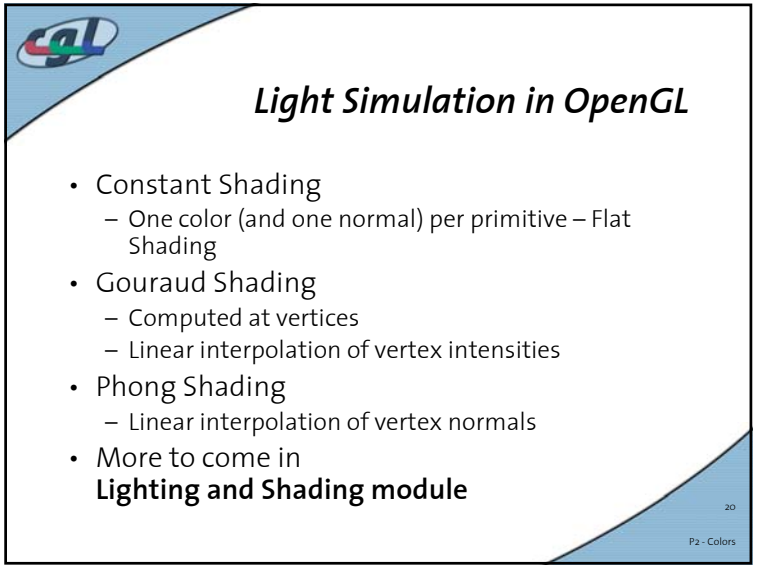






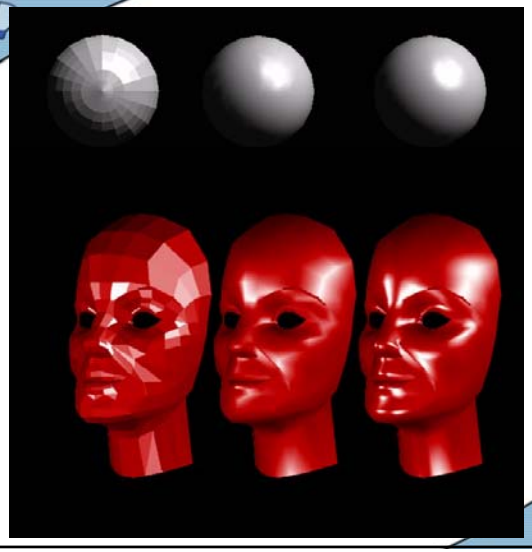
Principles of Lighting & Shading

- Lighting simulates how objects reflect light
  - material composition of object
  - light's color and position
  - global lighting parameters
    - ambient light
    - two sided lighting
  - available in both color index and RGBA mode



Light Simulation in OpenGL

- Constant Shading
  - One color (and one normal) per primitive – Flat Shading
- Gouraud Shading
  - Computed at vertices
  - Linear interpolation of vertex intensities
- Phong Shading
  - Linear interpolation of vertex normals
- More to come in **Lighting and Shading module**



Vertex Color Computation

- Lighting contributors
  - **Ambient** is color of the object from all the undirected light in a scene.
  - **Diffuse** is the base color of the object under current lighting. There must be a light shining on the object to get a diffuse contribution.
  - **Specular** is the contribution of the shiny highlights on the object.
  - **Emission** is the contribution added in if the object emits light (i.e. glows)

ambient

ambient + diffuse

ambient + diffuse
+ specular (phong)

Surface Normals

- Normals define how a surface reflects light

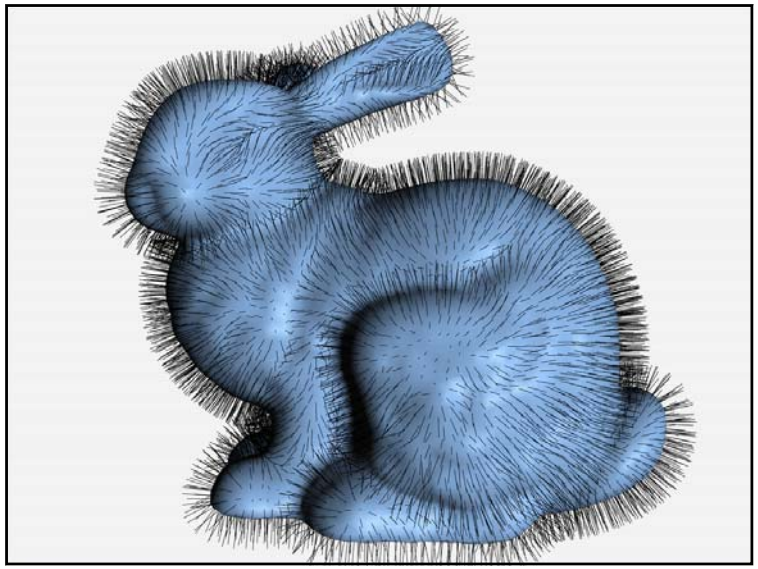
```
glNormal3f( x, y, z );
```

- Current normal is used to compute vertex's color
- Use unit normals for proper lighting scaling affects a normal's length

```
glEnable( GL_NORMALIZE );
```

or

```
glEnable( GL_RESCALE_NORMAL );
```



Material Properties

- Define the surfaces properties of a primitive

```
glMaterialfv(face, property, value);
```

- Separate materials for front and back

`GL_DIFFUSE`	Base color
`GL_SPECULAR`	Highlight Color
`GL_AMBIENT`	Low-light Color
`GL_EMISSION`	Glow Color
`GL_SHININESS`	Surface Smoothness

- Lights and their properties in another exercise

P1

- Goals
  - Color Spaces
  - Vertex Normals
  - Lighting and Shading in OpenGL
  - Display Lists

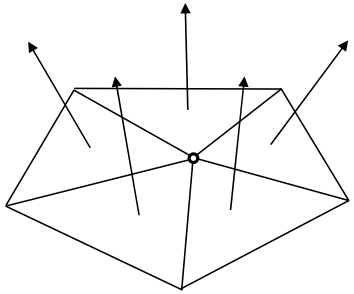
P1 – Data Structure

- First Array
  - Number of Nodes with corresponding 3D coordinates
  - “the vertices”
- Second Array
  - Number of Triangles with references to their vertices in first list
  - “the indices”

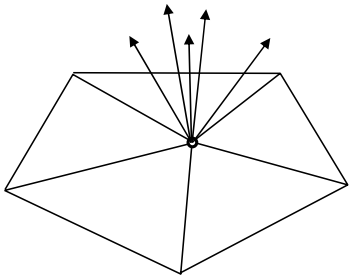
P1 – Normals

- Per-Triangle Normal
  - How to determine a normal of a triangle?
  - Vector algebra
- Per-Vertex Normal
  - Average of face normals

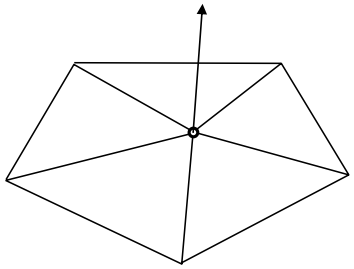
Vertex Normals



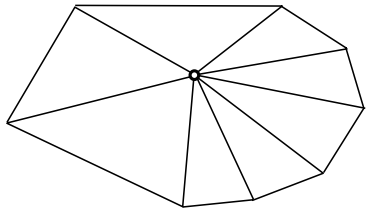
Vertex Normals



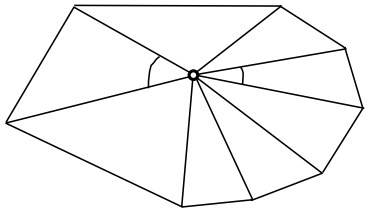
Vertex Normals



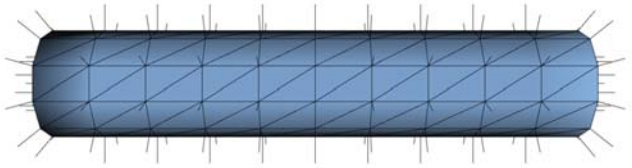
Vertex Normals

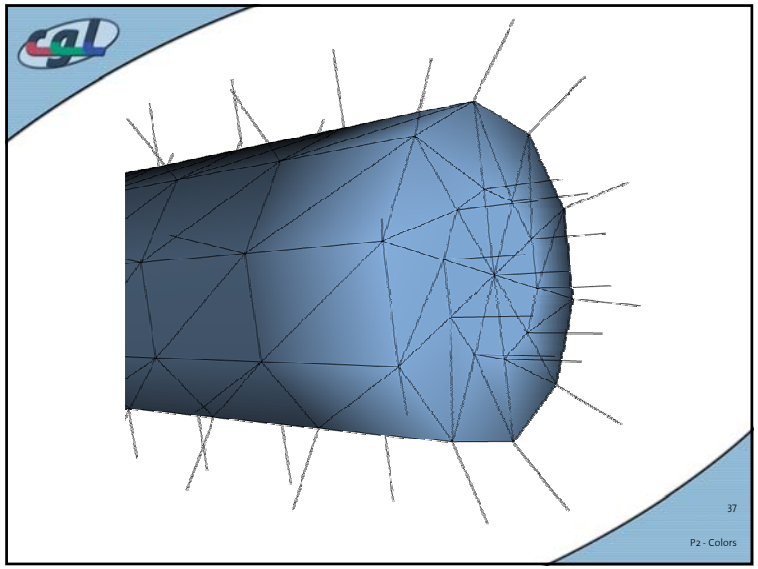


Vertex Normals



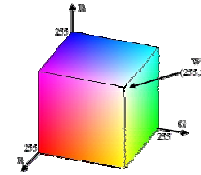
Better to weight normals by incident angle.



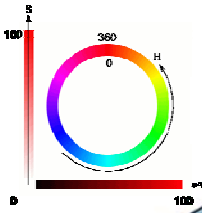


P1 – Color Spaces revisited

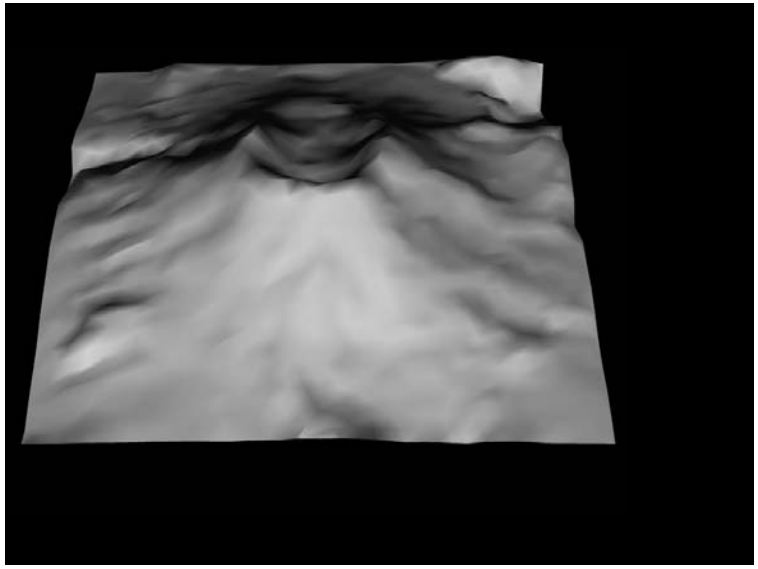
- RGB
  - Red , green and blue
  - Normalized to [0,1]
- HSV
  - Hue (dt. Farbton)
  - Saturation (dt. Sättigung)
  - Value (dt. Helligkeit)
  - Advantages of HSV
- HSV to RGB?
  - Hint: See script

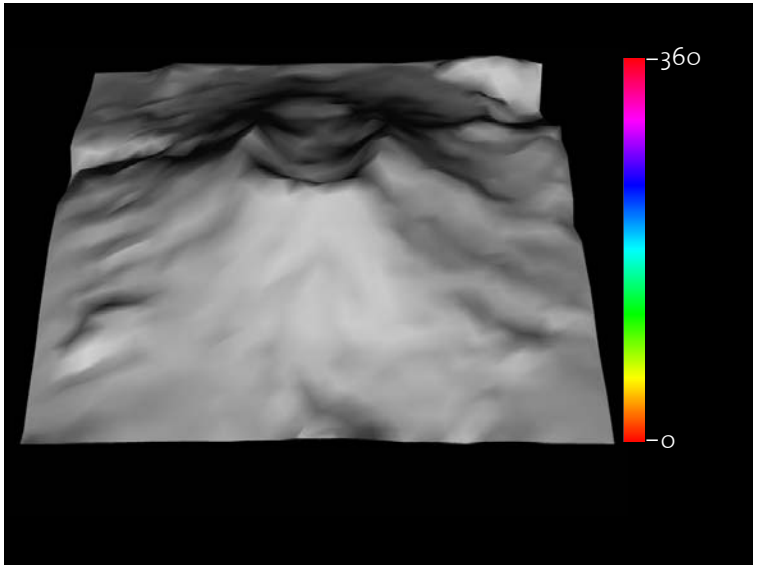


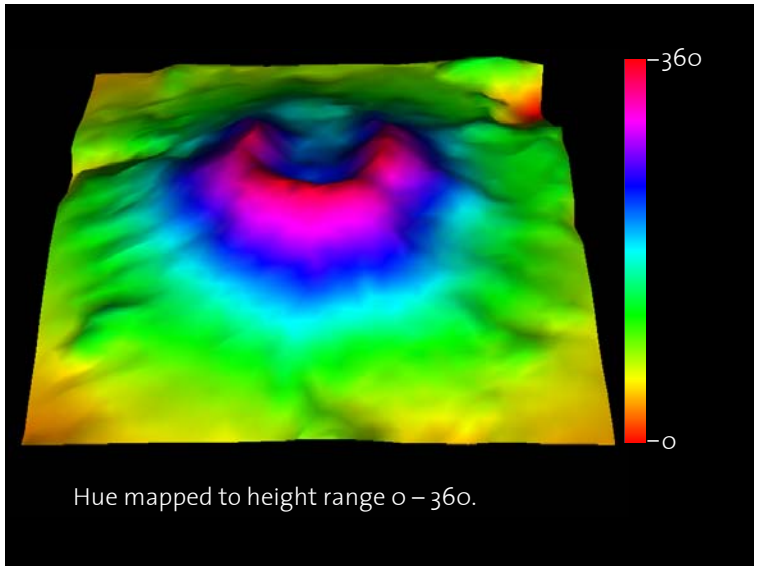
RGB



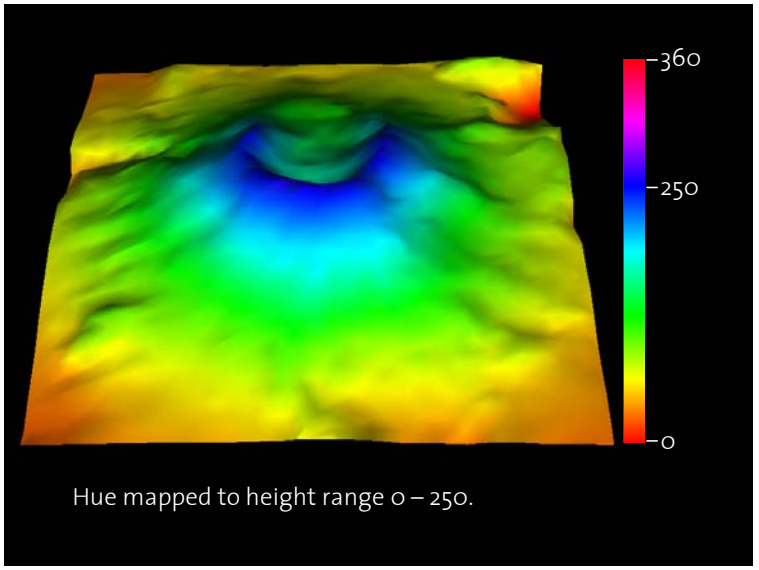
HSV



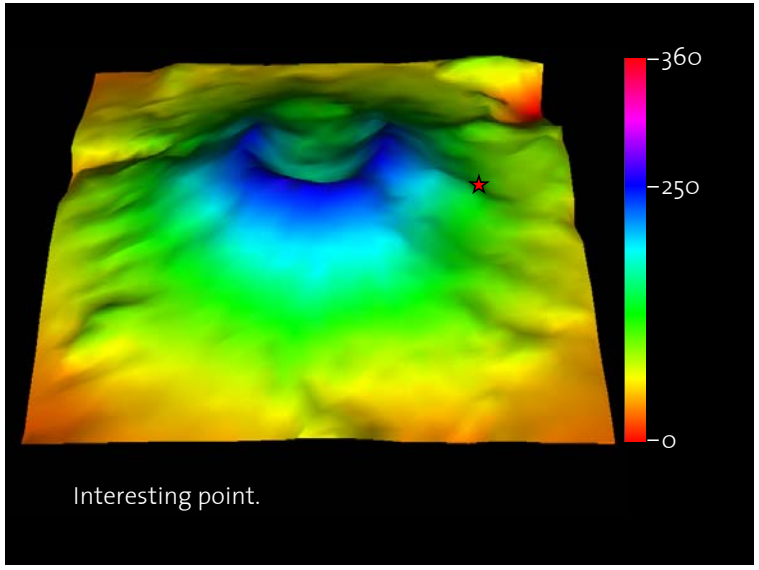




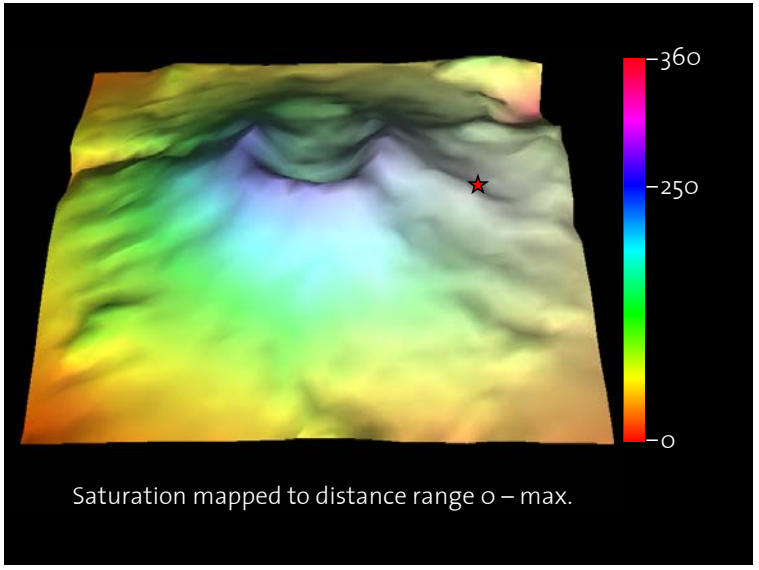
Hue mapped to height range 0 – 360.



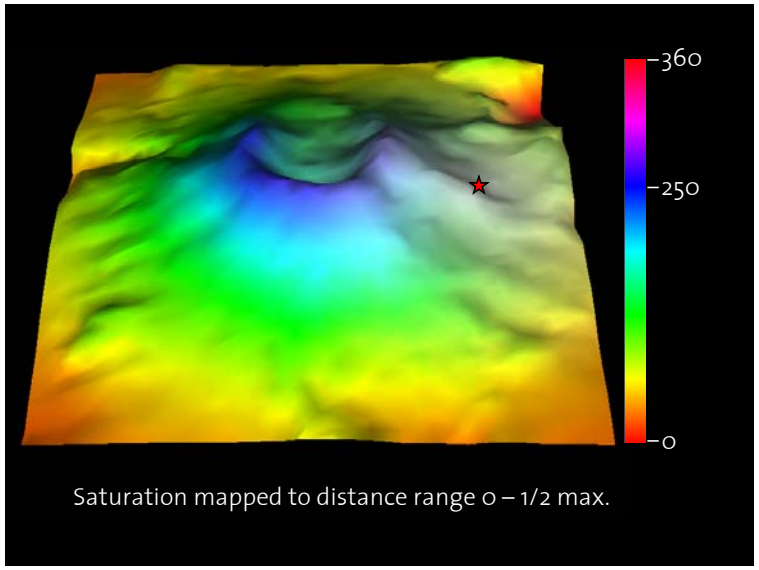
Hue mapped to height range 0 – 250.



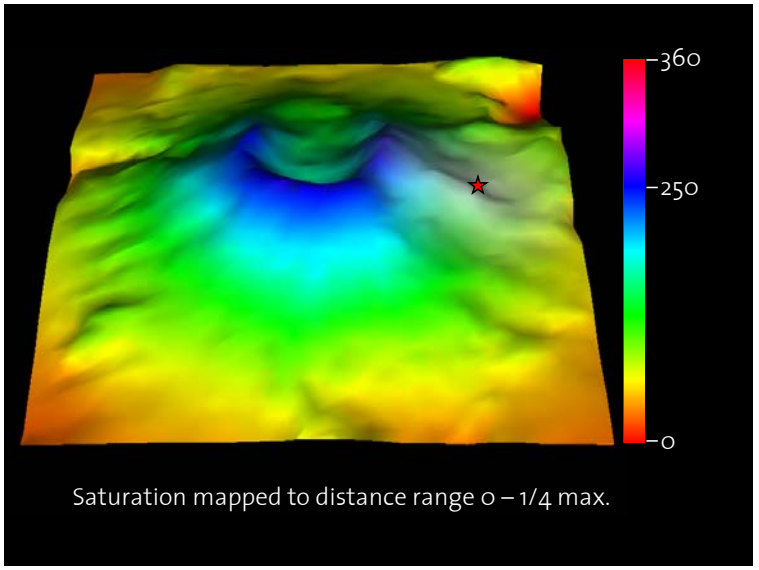
Interesting point.



Saturation mapped to distance range 0 – max.



Saturation mapped to distance range 0 – 1/2 max.



Saturation mapped to distance range 0 – 1/4 max.