Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Lighting and Shading II

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Objectives

- Continue discussion of shading
- Introduce modified Phong model
- Consider computation of required vectors

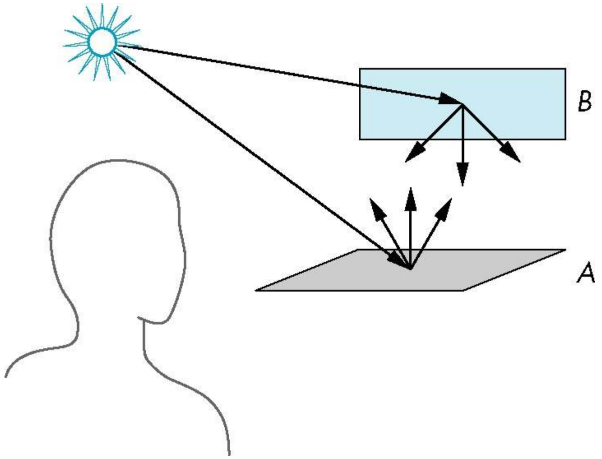
Ambient Light

- Ambient light is the result of multiple interactions between (large) light sources and the objects in the environment
- Amount and color depend on both the color of the light(s) and the material properties of the object
- Add k_a I_a to diffuse and specular terms

reflection coef   intensity of ambient light

Distance Terms

- The light from a point source that reaches a surface is inversely proportional to the square of the distance between them
- We can add a factor of the form $1/(a + bd + cd^2)$ to the diffuse and specular terms



- The constant and linear terms soften the effect of the point source

Light Sources

- In the Phong Model, we add the results from each light source
- Each light source has separate diffuse, specular, and ambient terms to allow for maximum flexibility even though this form does not have a physical justification
- Separate red, green and blue components
- Hence, 9 coefficients for each point source
  - I_{dr}, I_{dg}, I_{db}, I_{sr}, I_{sg}, I_{sb}, I_{ar}, I_{ag}, I_{ab}

Material Properties

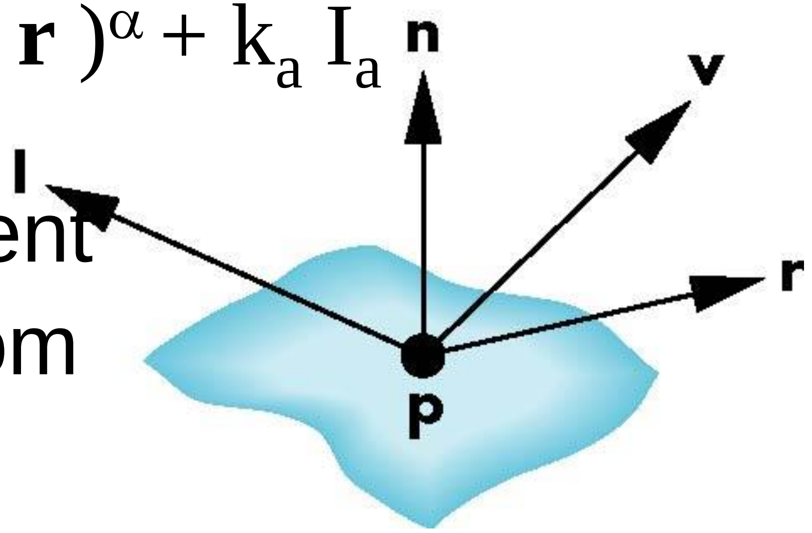
- Material properties match light source properties
  - Nine absorbtion coefficients
    - $k_{dr}, k_{dg}, k_{db}, k_{sr}, k_{sg}, k_{sb}, k_{ar}, k_{ag}, k_{ab}$
  - Shininess coefficient α

The University of New Mexico

Adding up the Components

For each light source and each color component, the Phong model can be written (without the distance terms) as

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^{\alpha} + k_a I_a$$



For each color component we add contributions from all sources

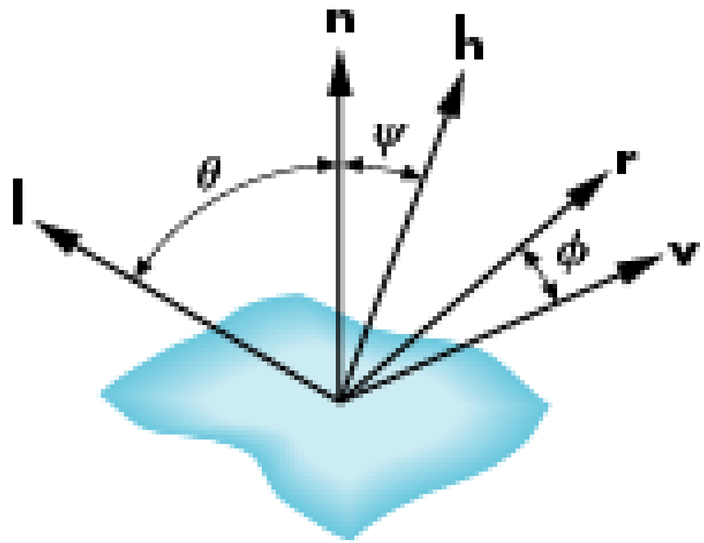
Modified Phong Model

- The specular term in the Phong model is problematic because it requires the calculation of a new reflection vector and view vector for each vertex
- Blinn suggested an approximation using the halfway vector that is more efficient

The Halfway Vector

- **h** is normalized vector halfway between **l** and **v**

$$\mathbf{h} = (\mathbf{l} + \mathbf{v})/\,|\,\mathbf{l} + \mathbf{v}\,|$$

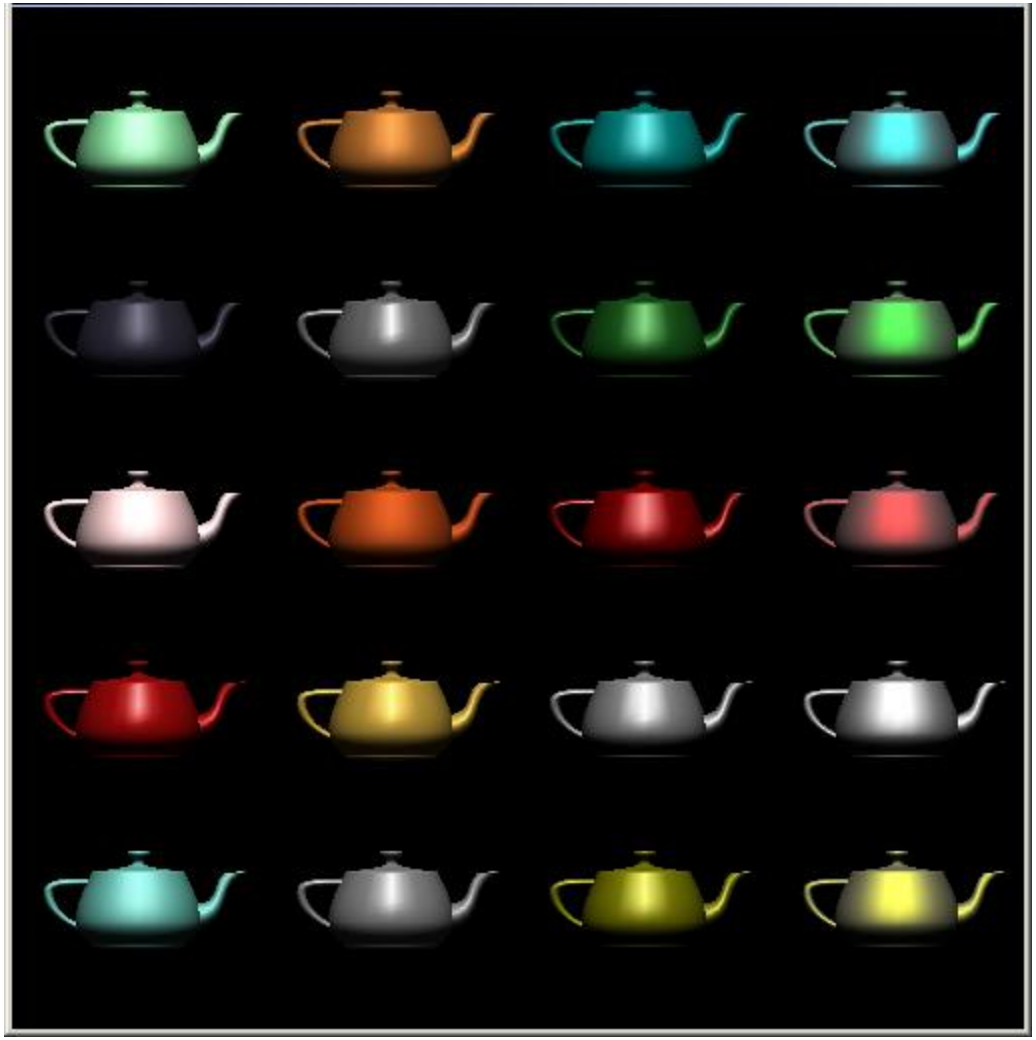


Using the halfway vector

- Replace $(\mathbf{v} \cdot \mathbf{r})^{\alpha}$ by $(\mathbf{n} \cdot \mathbf{h})^{\beta}$
- β is chosen to match shininess
- Note that halfway angle is half of angle between $\mathbf{r}$ and $\mathbf{v}$ if vectors are coplanar
- Resulting model is known as the modified Phong or Phong-Blinn lighting model
  - Specified in OpenGL standard

Example

Only differences in these teapots are the parameters in the modified Phong model



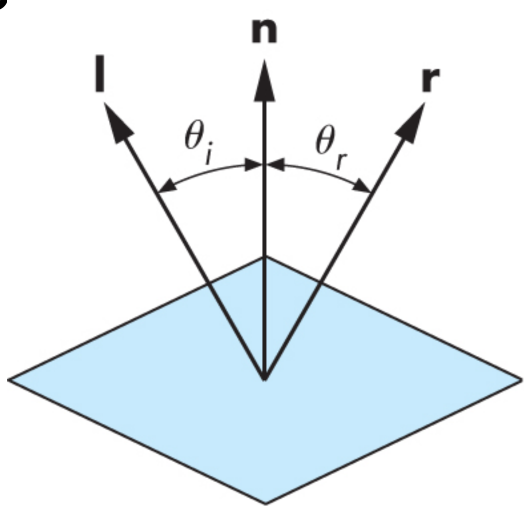
Computation of Vectors

- **l** and **v** are specified by the application
- Can computer **r** from **l** and **n**
- Problem is determining **n**
- For simple surfaces is can be determined but how we determine **n** differs depending on underlying representation of surface
- OpenGL leaves determination of normal to application
  - Exception for GLU quadrics and Bezier surfaces was deprecated

Computing Reflection Direction

- Angle of incidence = angle of reflection
- Normal, light direction and reflection direction are coplaner
- Want all three to be unit length

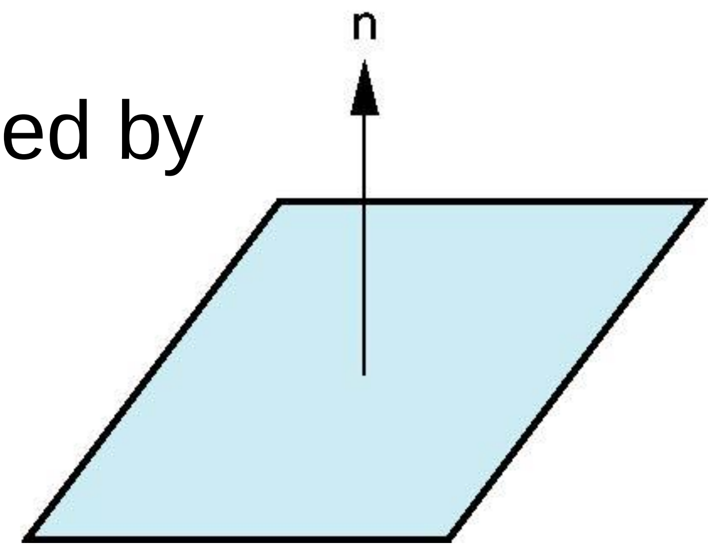
$$r = 2(l \bullet n)n - l$$



Plane Normals

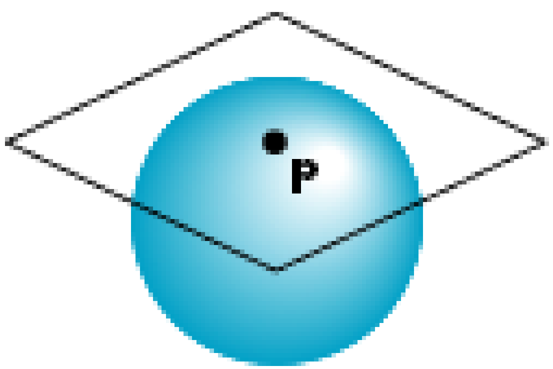
- Equation of plane: $ax+by+cz+d = 0$
- From Chapter 4 we know that plane is determined by three points p_0, p_2, p_3 or normal $\mathbf{n}$ and p_0
- Normal can be obtained by

$$\mathbf{n} = (p_2-p_0) \times (p_1-p_0)$$



Normal to Sphere

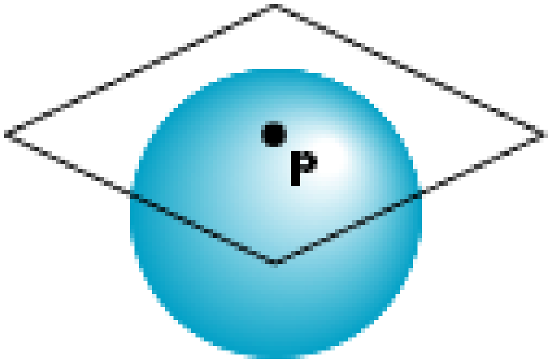
- Implicit function f(x,y.z)=0
- Normal given by gradient
- Sphere $f(\mathbf{p})=\mathbf{p}\cdot\mathbf{p}-1$
- $n = [\partial f/\partial x, \partial f/\partial y, \partial f/\partial z]^T=\mathbf{p}$



Parametric Form

- For sphere

$$x=x(u,v)=\cos u \sin v$$
$$y=y(u,v)=\cos u \cos v$$
$$z= z(u,v)=\sin u$$



- Tangent plane determined by vectors

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

- Normal given by cross product

$$\mathbf{n} = \partial \mathbf{p}/\partial u \times \partial \mathbf{p}/\partial v$$

General Case

- We can compute parametric normals for other simple cases
  - Quadrics
  - Parametric polynomial surfaces
    - Bezier surface patches (Chapter 11)

The University of New Mexico

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

The University of New Mexico

Lighting and Shading in WebGL

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Objectives

- Introduce the WebGL shading methods
  - Light and material functions on MV.js
  - per vertex vs per fragment shading
  - Where to carry out

WebGL lighting

- Need
  - Normals
  - Material properties
  - Lights
- State-based shading functions have been deprecated (glNormal, glMaterial, glLight)
- Compute in application or in shaders

Normalization

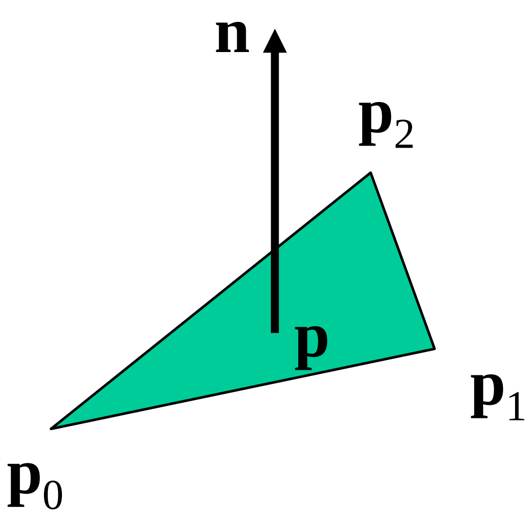
- Cosine terms in lighting calculations can be computed using dot product
- Unit length vectors simplify calculation
- Usually we want to set the magnitudes to have unit length but
  - Length can be affected by transformations
  - Note that scaling does not preserved length
- GLSL has a normalization function

Normal for Triangle

plane $\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$

$\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$

normalize $\mathbf{n} \leftarrow \mathbf{n}/|\mathbf{n}|$



Note that right-hand rule determines outward face

The University of New Mexico

Specifying a Point Light Source

- For each light source, we can set an RGBA for the diffuse, specular, and ambient components, and for the position

```
var diffuse0 = vec4(1.0, 0.0, 0.0, 1.0);
var ambient0 = vec4(1.0, 0.0, 0.0, 1.0);
var specular0 = vec4(1.0, 0.0, 0.0, 1.0);
var light0_pos = vec4(1.0, 2.0, 3,0, 1.0);
```

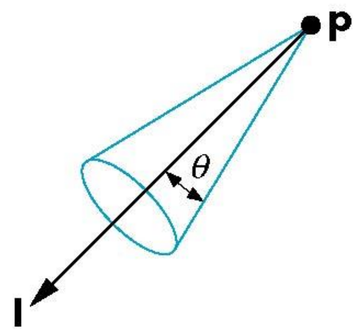
The University of New Mexico

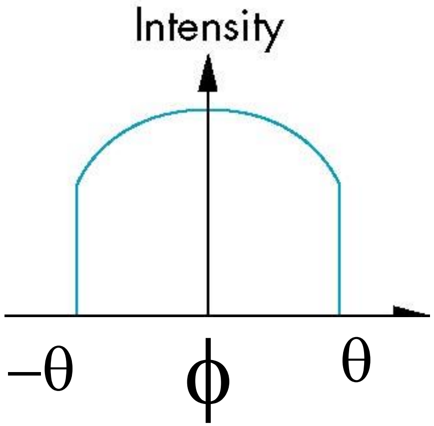
Distance and Direction

- The source colors are specified in RGBA
- The position is given in homogeneous coordinates
  - If w =1.0, we are specifying a finite location
  - If w =0.0, we are specifying a parallel source with the given direction vector
- The coefficients in distance terms are usually quadratic (1/(a+b*d+c*d*d)) where d is the distance from the point being rendered to the light source

Spotlights

- Derive from point source
  - Direction
  - Cutoff
  - Attenuation Proportional to $\cos^{\alpha}\phi$





Global Ambient Light

- Ambient light depends on color of light sources
  - A red light in a white room will cause a red ambient term that disappears when the light is turned off
- A global ambient term that is often helpful for testing

Moving Light Sources

- Light sources are geometric objects whose positions or directions are affected by the model-view matrix
- Depending on where we place the position (direction) setting function, we can
  - Move the light source(s) with the object(s)
  - Fix the object(s) and move the light source(s)
  - Fix the light source(s) and move the object(s)
  - Move the light source(s) and object(s) independently

Light Properties

```
var lightPosition = vec4(1.0, 1.0, 1.0, 0.0 );
var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse = vec4( 1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4( 1.0, 1.0, 1.0, 1.0 );
```

Material Properties

- Material properties should match the terms in the light model
- Reflectivities
- w component gives opacity

```
var materialAmbient = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse = vec4( 1.0, 0.8, 0.0, 1.0);
var materialSpecular = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;
```

Using MV.js for Products

```
var ambientProduct = mult(lightAmbient, materialAmbient);
var diffuseProduct = mult(lightDiffuse, materialDiffuse);
var specularProduct = mult(lightSpecular, materialSpecular);
 gl.uniform4fv(gl.getUniformLocation(program,
                "ambientProduct"),       flatten(ambientProduct));
 gl.uniform4fv(gl.getUniformLocation(program,
                "diffuseProduct"),       flatten(diffuseProduct) );
 gl.uniform4fv(gl.getUniformLocation(program,
                 "specularProduct"),       flatten(specularProduct) );
 gl.uniform4fv(gl.getUniformLocation(program,
                 "lightPosition"),       flatten(lightPosition) );
gl.uniform1f(gl.getUniformLocation(program,
                "shininess"),materialShininess);
```

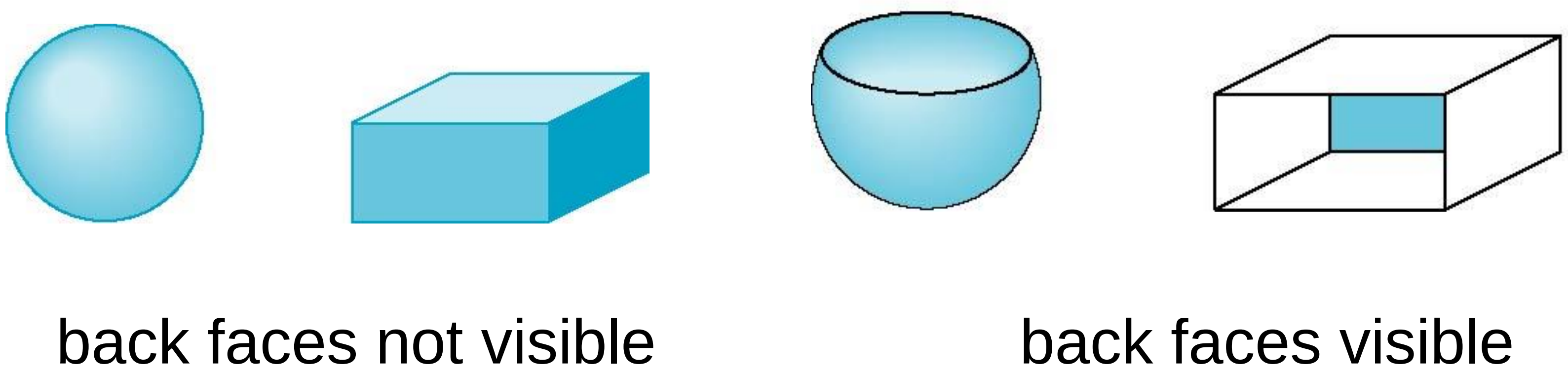
Adding Normals for Quads

```
function quad(a, b, c, d) {
    var t1 = subtract(vertices[b], vertices[a]);
    var t2 = subtract(vertices[c], vertices[b]);
    var normal = cross(t1, t2);
    var normal = vec3(normal);
    normal = normalize(normal);

    pointsArray.push(vertices[a]);
    normalsArray.push(normal);
      .
      .
      .
```

Front and Back Faces

- Every face has a front and back
- For many objects, we never see the back face so we don't care how or if it's rendered
- If it matters, we can handle in shader



Emissive Term

- We can simulate a light source in WebGL by giving a material an emissive component
- This component is unaffected by any sources or transformations

Transparency

- Material properties are specified as RGBA values
- The A value can be used to make the surface translucent
- The default is that all surfaces are opaque
- Later we will enable blending and use this feature
- However with the HTML5 canvas, A<1 will mute colors

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Polygonal Shading

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Polygonal Shading

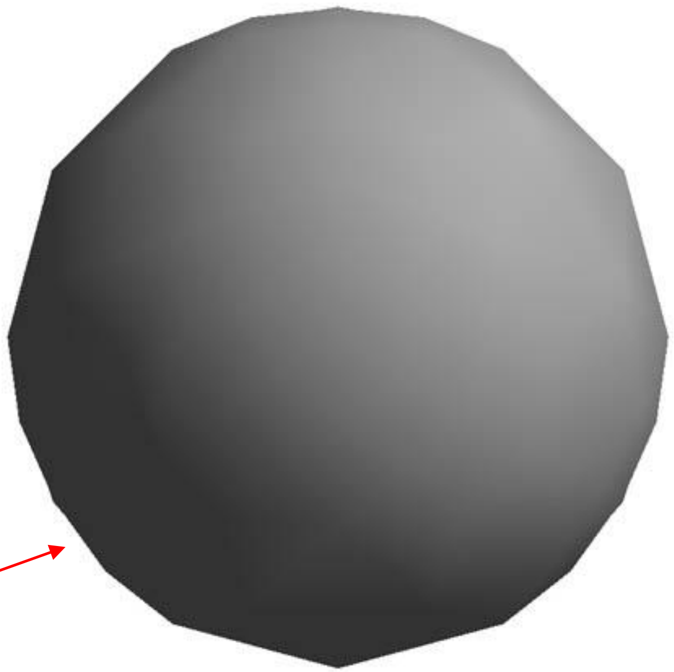
- In per vertex shading, shading calculations are done for each vertex
  - Vertex colors become vertex shades and can be sent to the vertex shader as a vertex attribute
  - Alternately, we can send the parameters to the vertex shader and have it compute the shade
- By default, vertex shades are interpolated across an object if passed to the fragment shader as a varying variable (smooth shading)
- We can also use uniform variables to shade with a single shade (flat shading)

Polygon Normals

- Triangles have a single normal
  - Shades at the vertices as computed by the modified Phong model can be almost same
  - Identical for a distant viewer (default) or if there is no specular component
- Consider model of sphere
- Want different normals at each vertex even though this concept is not quite correct mathematically

Smooth Shading

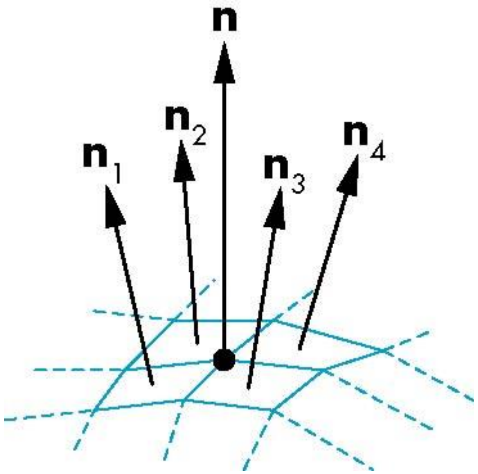
- We can set a new normal at each vertex
- Easy for sphere model
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Now smooth shading works
- Note *silhouette edge*



Mesh Shading

- The previous example is not general because we knew the normal at each vertex analytically
- For polygonal models, Gouraud proposed we use the average of the normals around a mesh vertex

$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4)/\ |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$$



Gouraud and Phong Shading

- Gouraud Shading
  - Find average normal at each vertex (vertex normals)
  - Apply modified Phong model at each vertex
  - Interpolate vertex shades across each polygon
- Phong shading
  - Find vertex normals
  - Interpolate vertex normals across edges
  - Interpolate edge normals across polygon
  - Apply modified Phong model at each fragment

Comparison

- If the polygon mesh approximates surfaces with a high curvatures, Phong shading may look smooth while Gouraud shading may show edges
- Phong shading requires much more work than Gouraud shading
  - Until recently not available in real time systems
  - Now can be done using fragment shaders
- Both need data structures to represent meshes so we can obtain vertex normals

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Per Vertex and Per Fragment Shaders

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Vertex Lighting Shaders I

```
// vertex shader

attribute vec4 vPosition;
attribute vec4 vNormal;
varying vec4 fColor;
uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float shininess;

void main()
{
```

Vertex Lighting Shaders II

```
vec3 pos = -(modelViewMatrix * vPosition).xyz;
 vec3 light = lightPosition.xyz;
 vec3 L = normalize( light - pos );
 vec3 E = normalize( -pos );
 vec3 H = normalize( L + E );

 // Transform vertex normal into eye coordinates

          vec3 N = normalize( (modelViewMatrix*vNormal).xyz);

 // Compute terms in the illumination equation
```

Vertex Lighting Shaders III

```
// Compute terms in the illumination equation
    vec4 ambient = AmbientProduct;

    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*DiffuseProduct;
    float Ks = pow( max(dot(N, H), 0.0), Shininess );
    vec4  specular = Ks * SpecularProduct;
    if( dot(L, N) < 0.0 )  specular = vec4(0.0, 0.0, 0.0, 1.0);
    gl_Position = Projection * ModelView * vPosition;

    fColor = ambient + diffuse + specular;
    fColor.a = 1.0;
}
```

Vertex Lighting Shaders IV

```
// fragment shader

precision mediump float;

varying vec4 fColor;

voidmain()
{
   gl_FragColor = fColor;
}
```

Fragment Lighting Shaders I

```
// vertex shader

attribute vec4 vPosition;
attribute vec4 vNormal;
varying vec3 N, L, E;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
```

Fragment Lighting Shaders II

```
void main()
{
    vec3 pos = -(modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    L = normalize( light - pos );
    E =  -pos;
    N = normalize( (modelViewMatrix*vNormal).xyz);
    gl_Position = projectionMatrix * modelViewMatrix * vPosition;
    };
```

Fragment Lighting Shaders III

```
// fragment shader

precision mediump float;

uniform vec4 ambientProduct;
uniform vec4 diffuseProduct;
uniform vec4 specularProduct;
uniform float shininess;
varying vec3 N, L, E;

void main()
{
```

Fragment Lighting Shaders IV

```
    vec4 fColor;
    vec3 H = normalize( L + E );
    vec4 ambient = ambientProduct;
    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*diffuseProduct;
    float Ks = pow( max(dot(N, H), 0.0), shininess );
    vec4  specular = Ks * specularProduct;
       if( dot(L, N) < 0.0 ) specular = vec4(0.0, 0.0, 0.0, 1.0);
    fColor = ambient + diffuse +specular;
    fColor.a = 1.0;
    gl_FragColor = fColor;
}
```

Teapot Examples

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Marching Squares

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Nontrivial two-dimensional application
- Important method for
  - Contour plots
  - Implicit function visualization
- Extends to important method for volume visualization
- This lecture is optional but should be interesting to most of you

Displaying Implicit Functions

- Consider the implicit function

$$g(x,y)=0$$

- Given an x, we cannot in general find a corresponding y
- Given an x and a y, we can test if they are on the curve

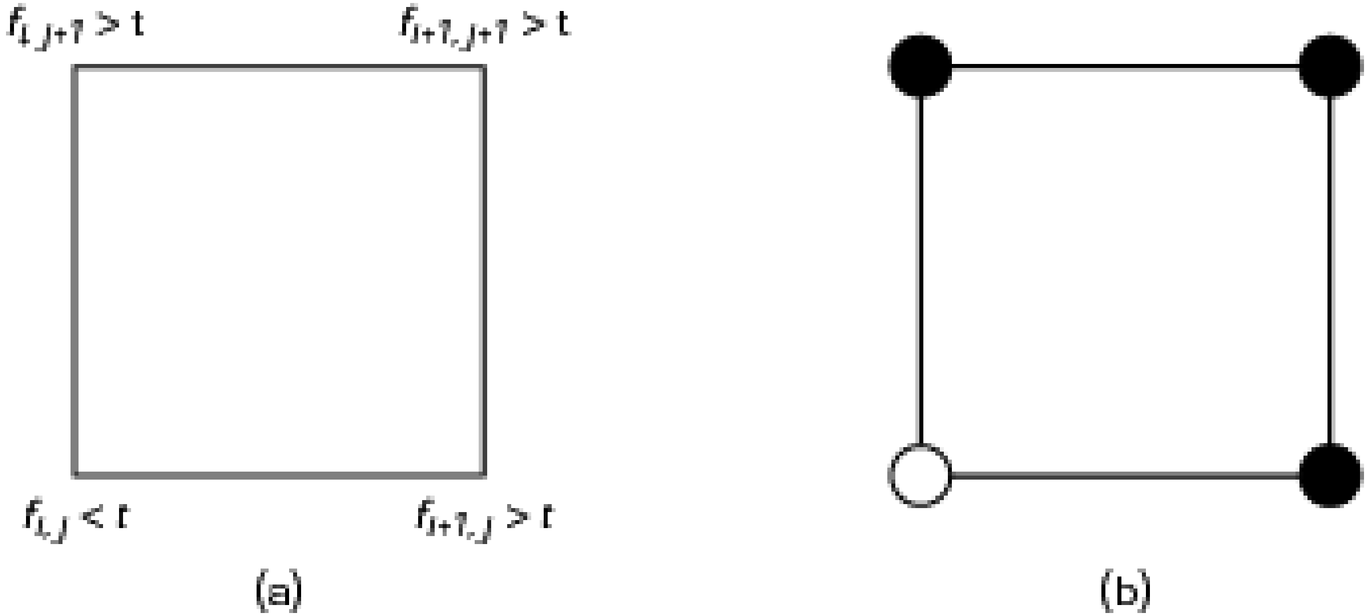
Height Fields and Contours

- In many applications, we have the heights given by a function of the form $z=f(x,y)$
- To find all the points that have a given height t, we have to solve the implicit equation $g(x,y)=f(x,y)-t=0$
- Such a function determines the **isocurves** or **contours** of f for the **isovalue** t

Marching Squares

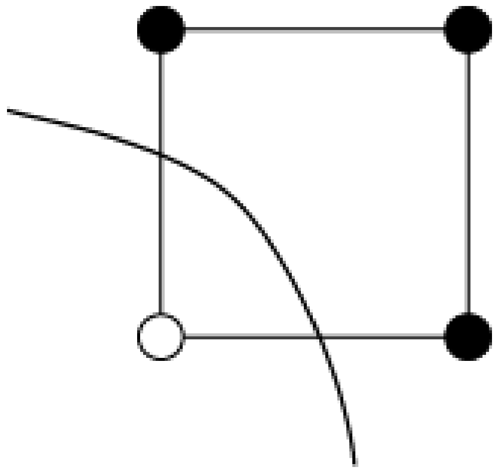
- Displays isocurves or contours for functions $f(x,y) = t$
- Sample $f(x,y)$ on a regular grid yielding samples $\{f_{ij}(x,y)\}$
- These samples can be greater than, less than, or equal to t
- Consider four samples $f_{ij}(x,y)$, $f_{i+1,j}(x,y)$, $f_{i+1,j+1}(x,y)$, $f_{i,j+1}(x,y)$
- These samples correspond to the corners of a cell
- Color the corners by whether they exceed or are less than the contour value t

Cells and Coloring

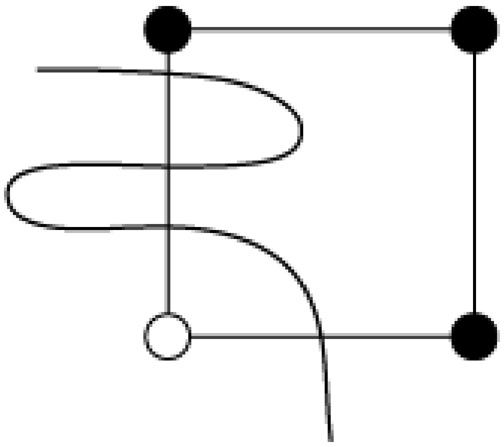


Occam's Razor

- Contour must intersect edge between a black and white vertex an odd number of times
- Pick simplest interpretation: one crossing

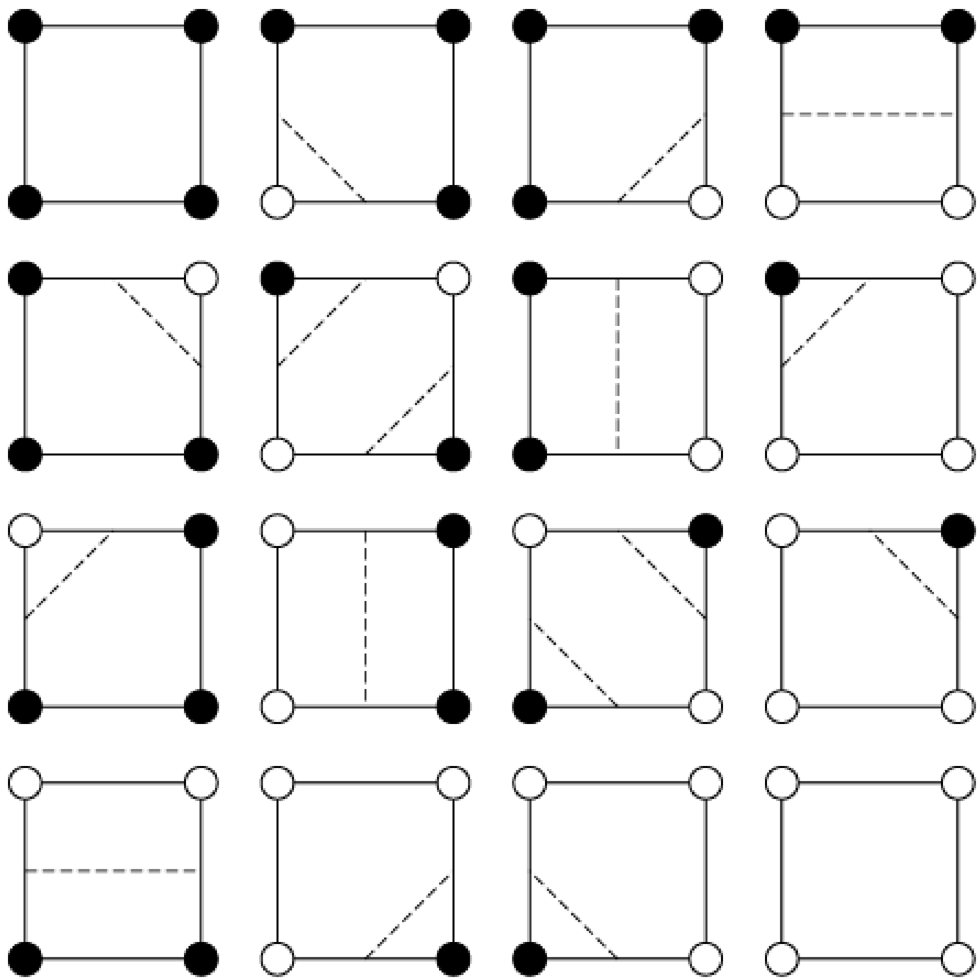


(a)



(b)

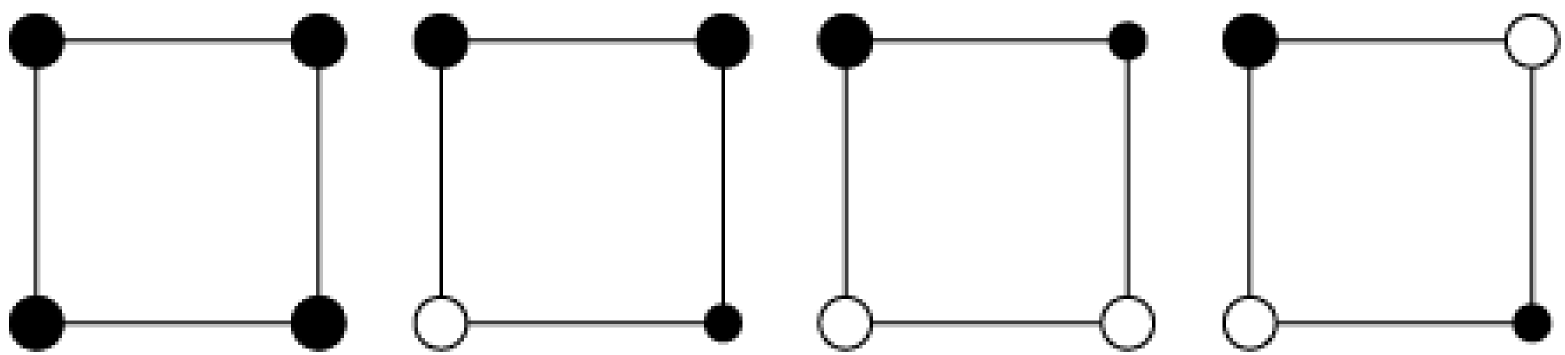
16 Cases



The University of New Mexico

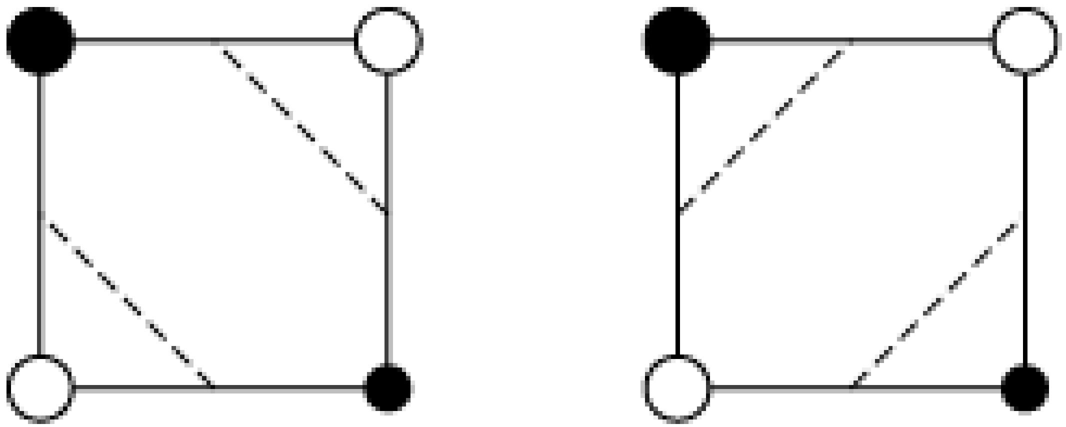
Unique Cases

- Taking out rotational and color swapping symmetries leaves four unique cases
- First three have a simple interpretation



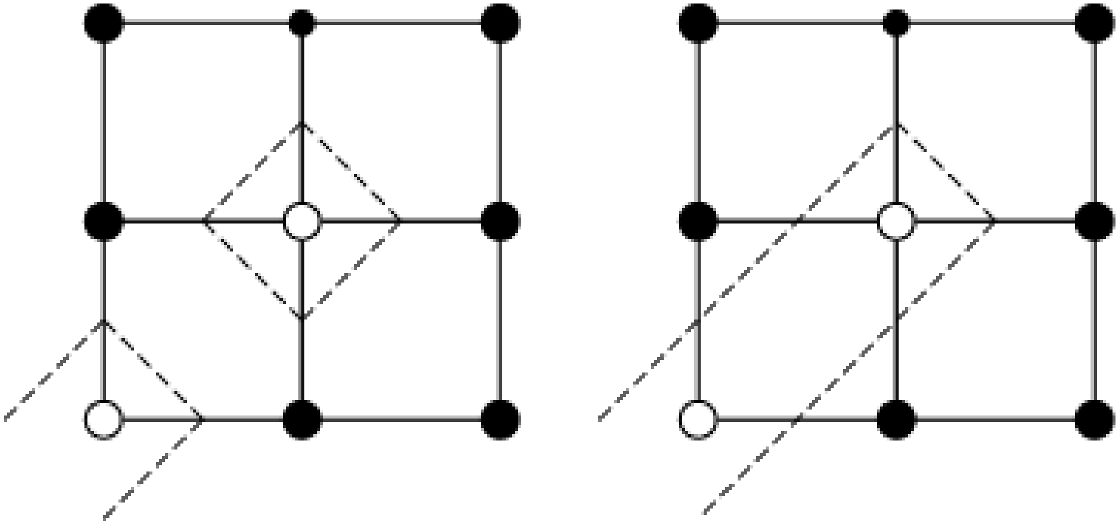
Ambiguity Problem

- Diagonally opposite cases have two equally simple possible interpretations

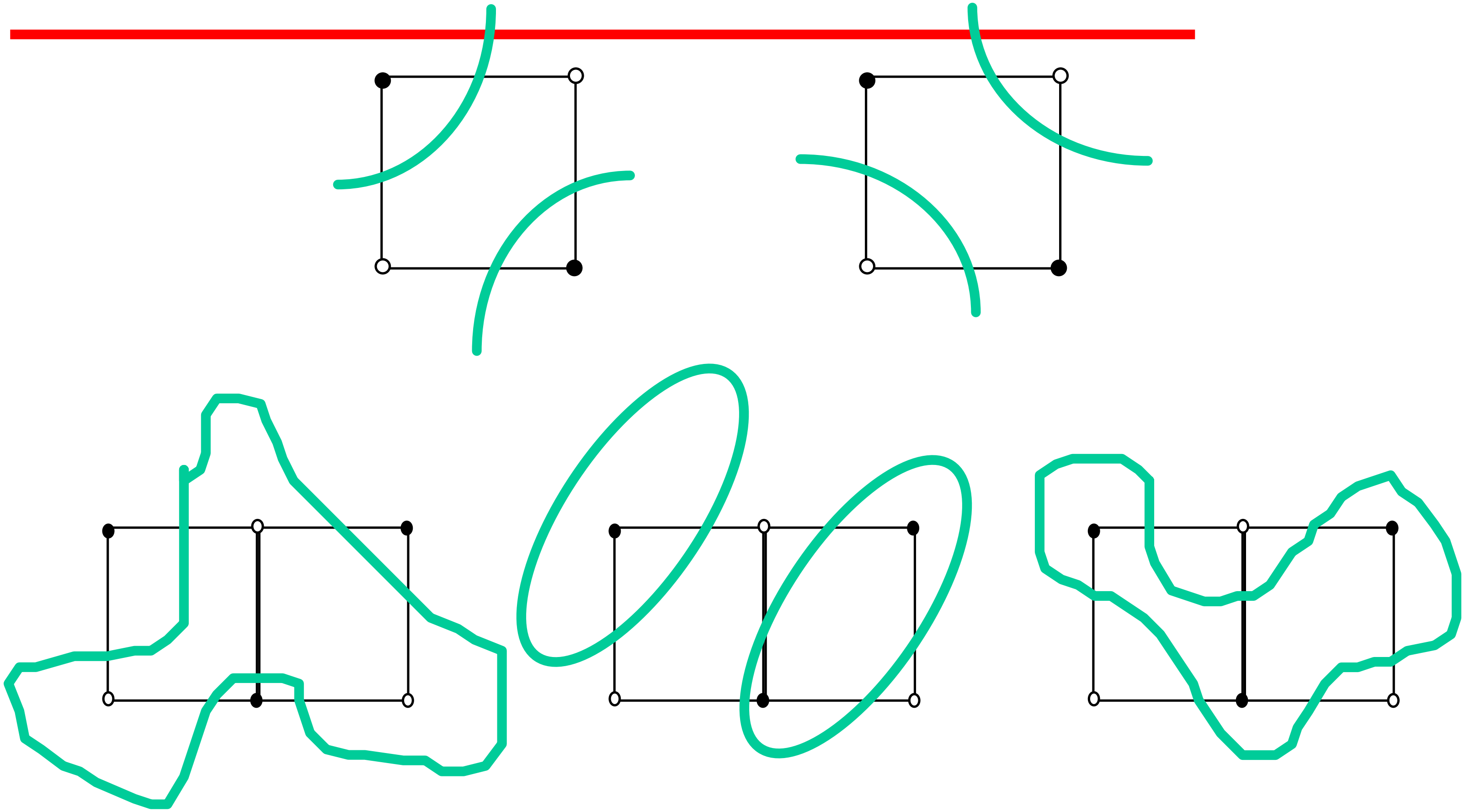


Ambiguity Example

- Two different possibilities below
- More possibilities on next slide



Ambiguity Problem

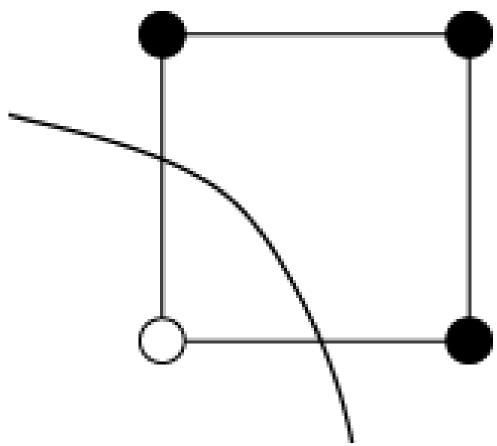


Is Problem Resolvable?

- Problem is a sampling problem
  - Not enough samples to know the local detail
  - No solution in a mathematical sense without extra information
- More of a problem with volume extension (marching cubes) where selecting “wrong” interpretation can leave a hole in a surface
- Multiple methods in literature to give better appearance
  - Supersampling
  - Look at larger area before deciding

Interpolating Edges

- We can compute where contour intersects edge in multiple ways
  - Halfway between vertics
  - Interpolated based on difference between contour value and value at vertices

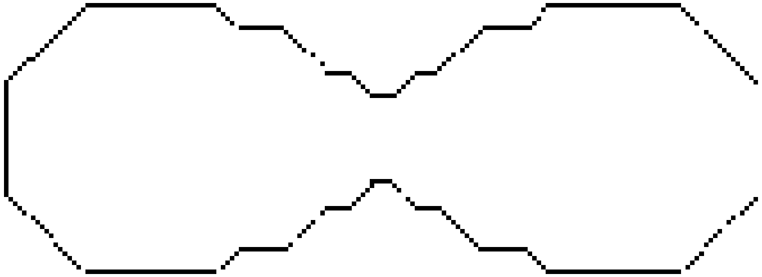


Example: Oval of Cassini

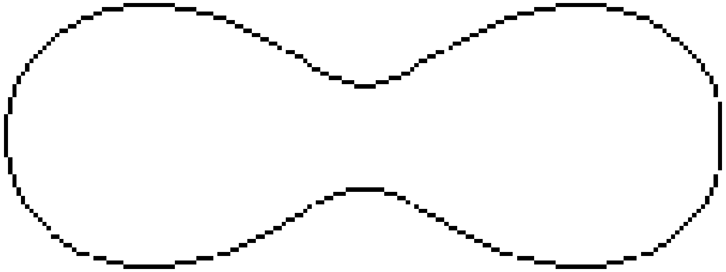
$$f(x,y)=(x^2+y^2+a^2)^2-4a^2x^2-b^4$$

Depending on a and b we can have 0, 1, or 2 curves

midpoint intersections

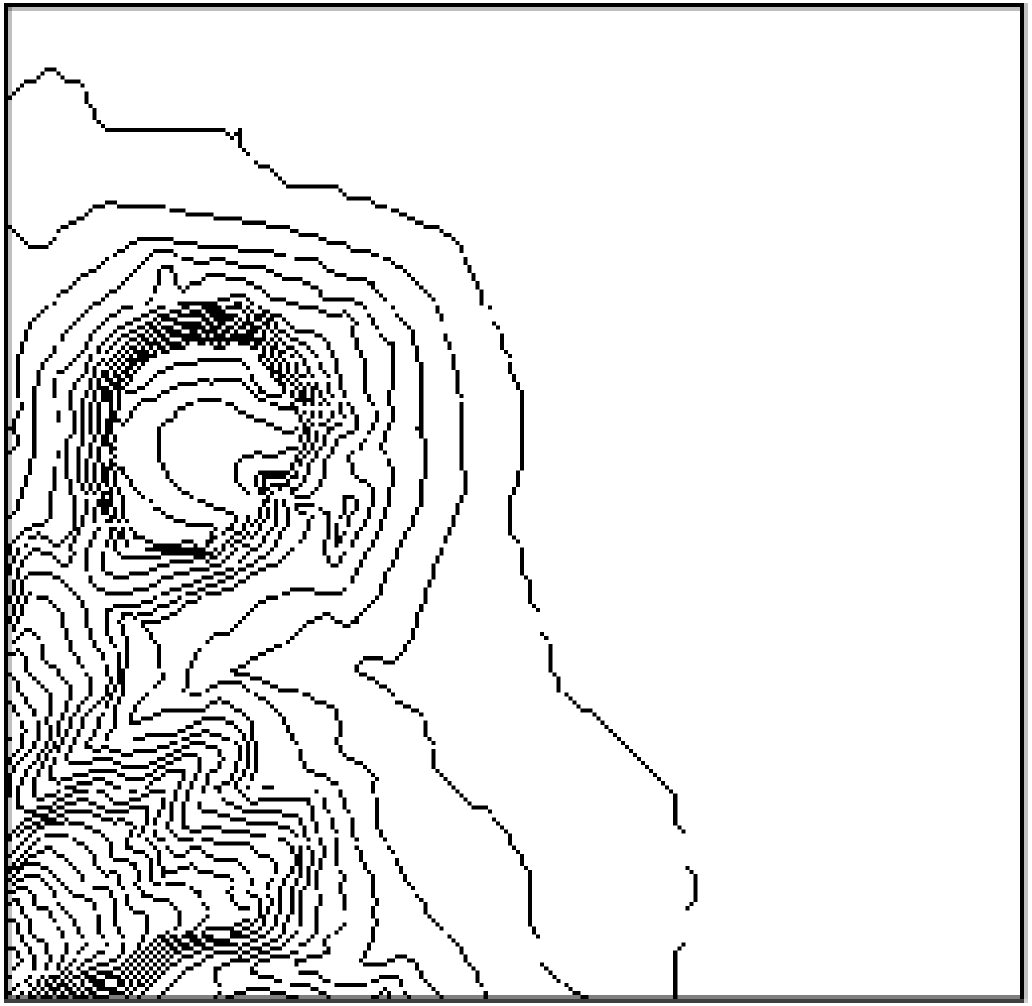


interpolating intersections



Contour Map

- Diamond Head, Oahu Hawaii
- Shows contours for many contour values



Marching Cubes

- Isosurface: solution of $g(x,y,z)=c$
- Use same argument to derive method but with a cubic cell (8 vertices, 256 colorings)
- Standard method of volume visualization
- Suggested by Lorensen and Kline before marching squares
- Note inherent parallelism of both marching cubes and marching squares