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)

The University of New Mexico

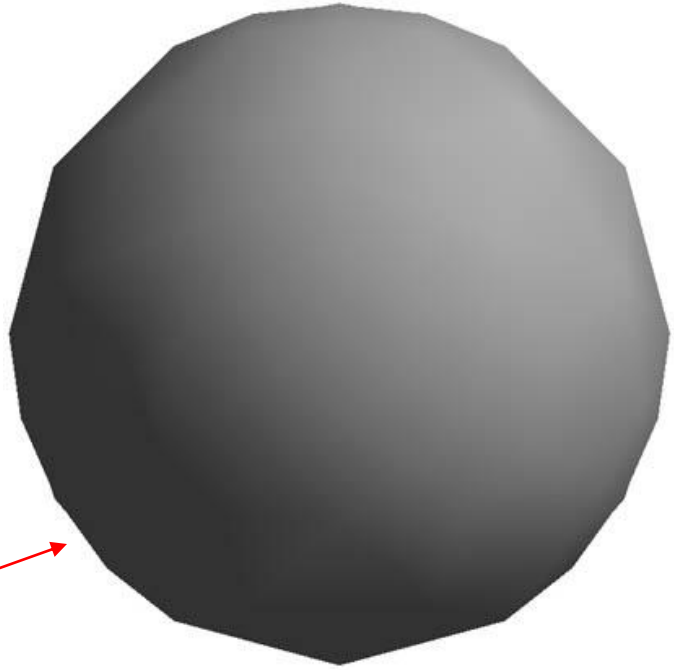
Polygon Normals

- 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

The University of New Mexico

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*

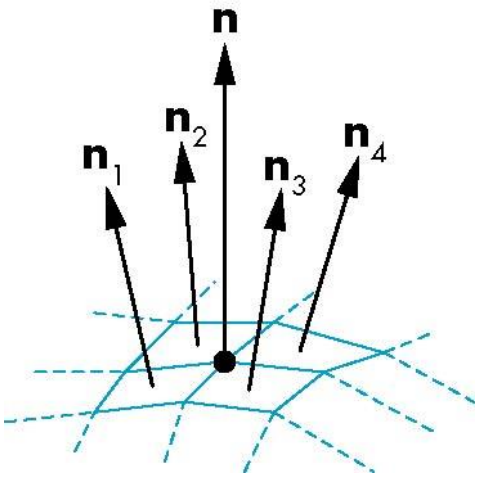


The University of New Mexico

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

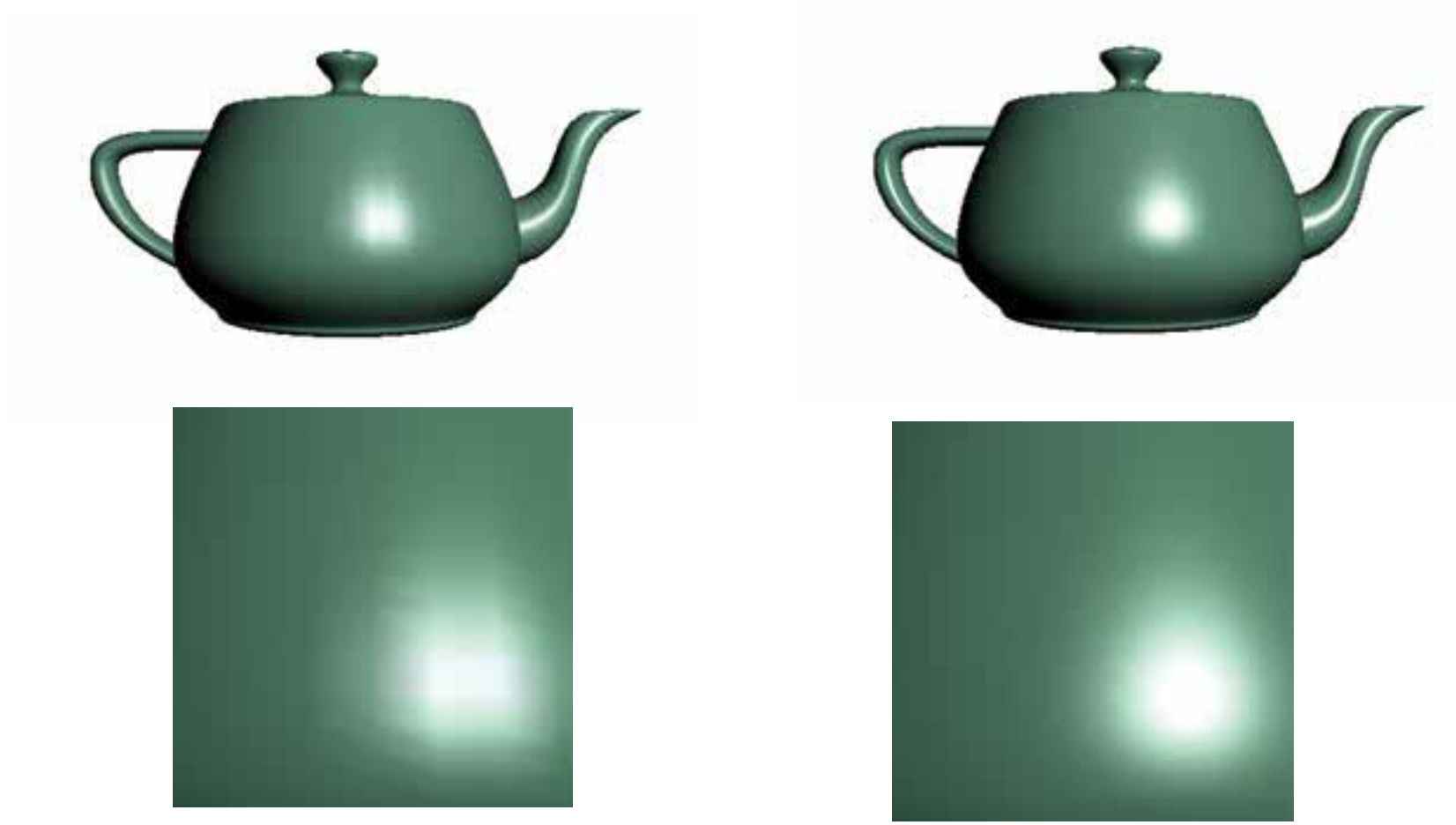
```
// fragment shader

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;
}
```

The University of New Mexico

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

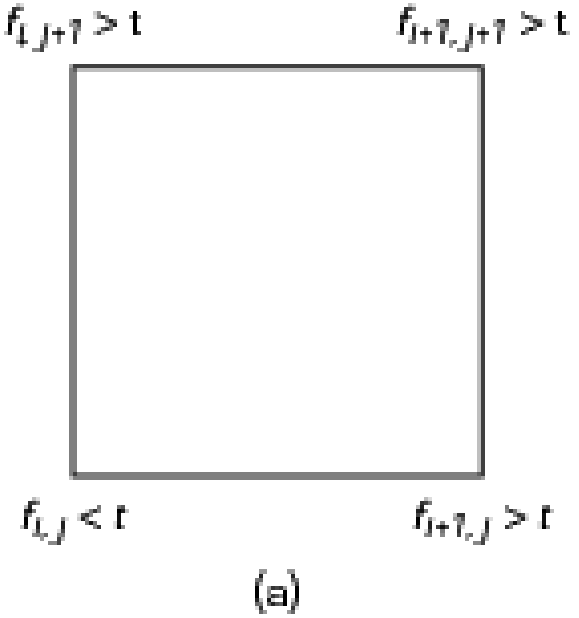
Height Fields and Contours

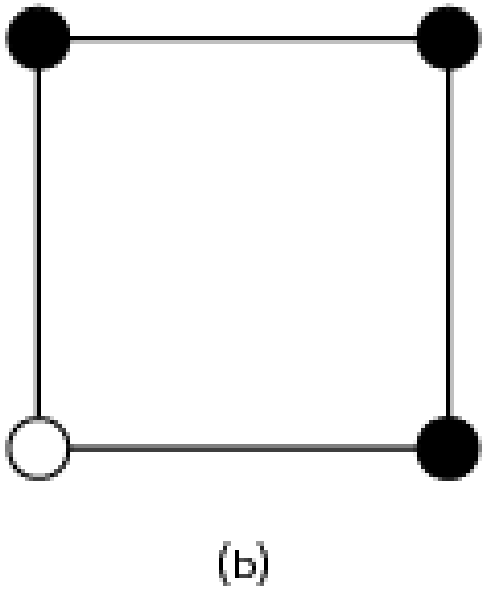
- In many applications, we have the heights given by a function of the form $z=f(x,y)$
- To find all the points that have a given height t, we have to solve the implicit equation $g(x,y)=f(x,y)-t=0$
- Such a function determines the **isocurves** or **contours** of f for the **isovalue** t

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

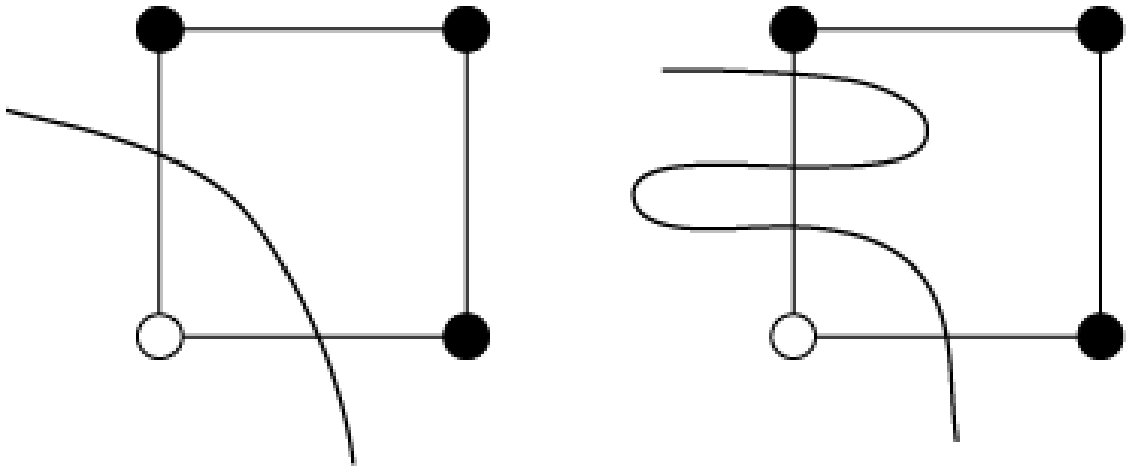




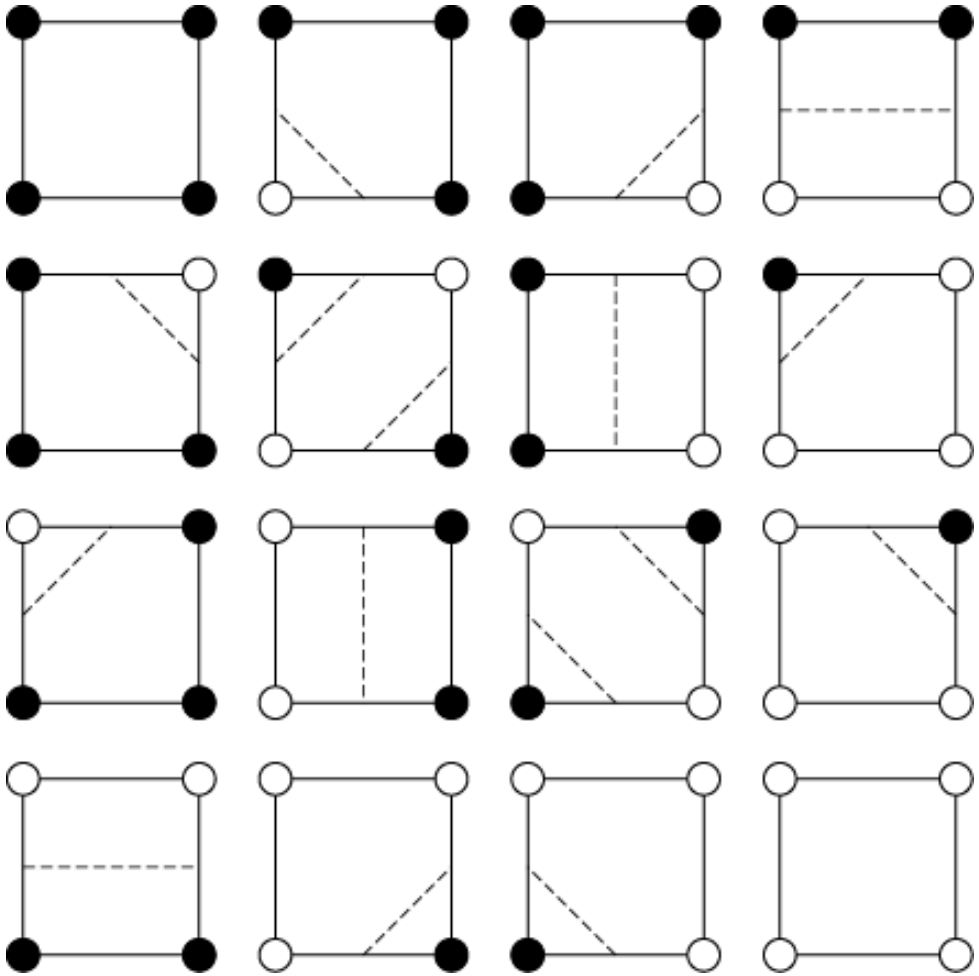
The University of New Mexico

Occam's Razor

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

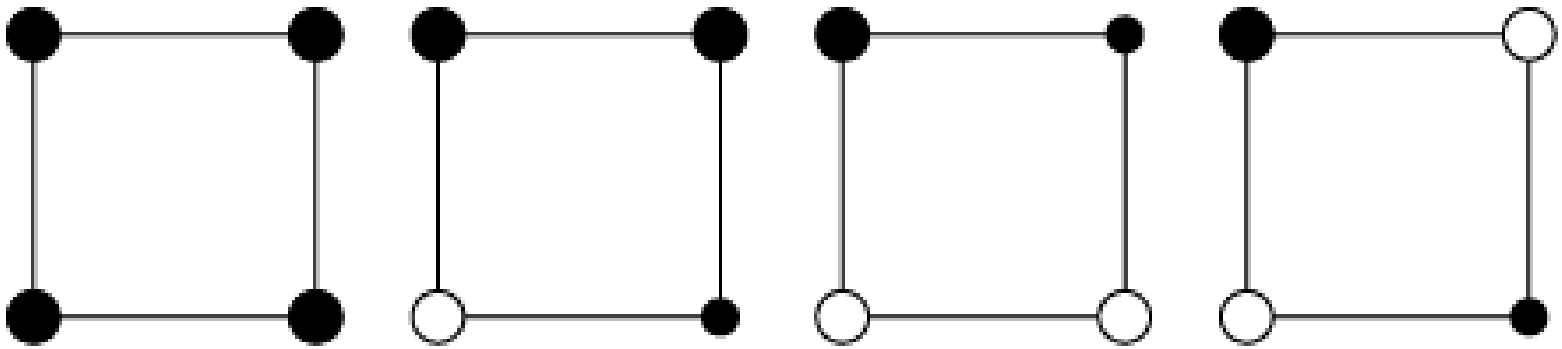


16 Cases



Unique Cases

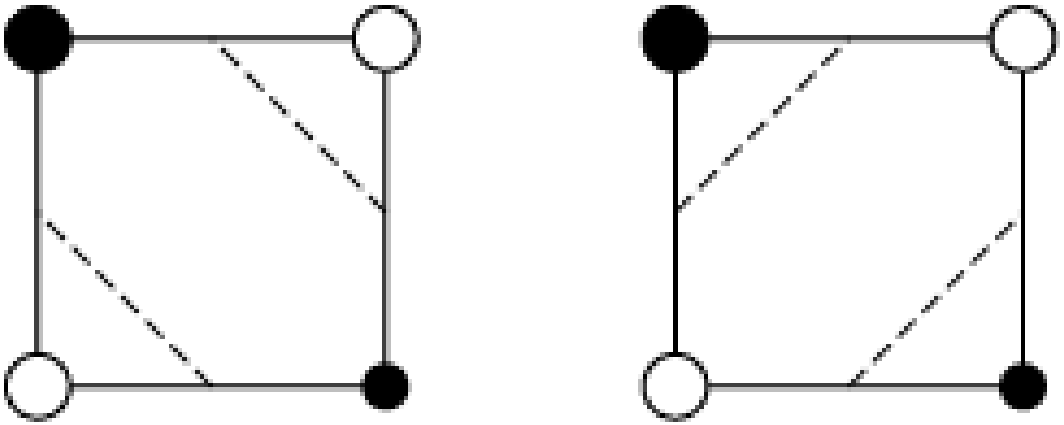
- Taking out rotational and color swapping symmetries leaves four unique cases
- First three have a simple interpretation



The University of New Mexico

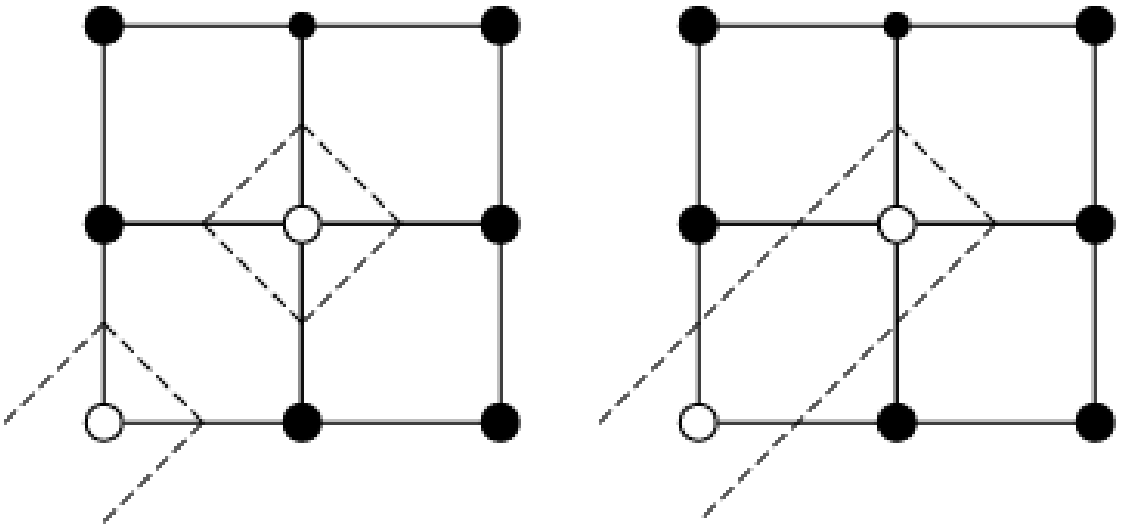
Ambiguity Problem

- Diagonally opposite cases have two equally simple possible interpretations

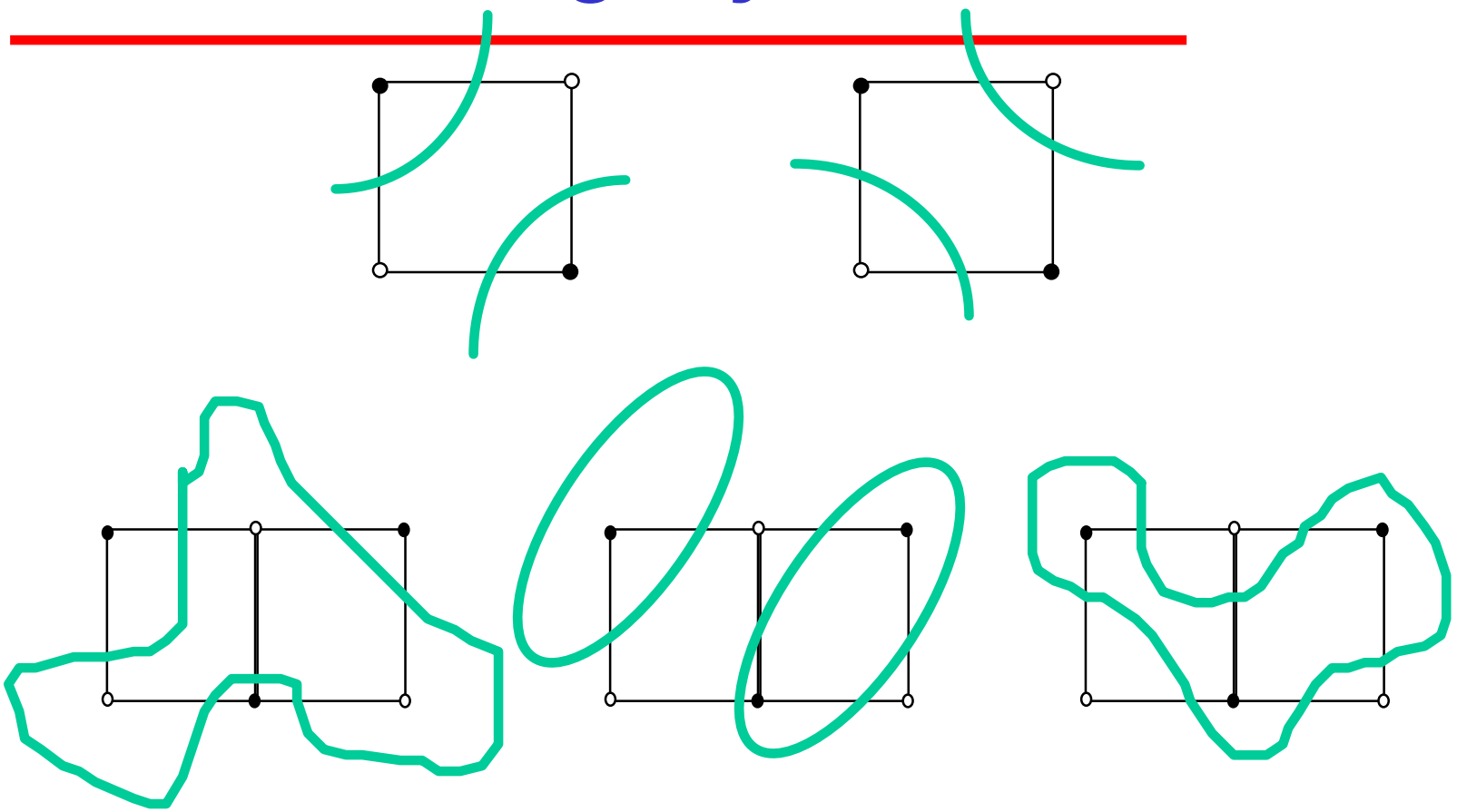


Ambiguity Example

- Two different possibilities below
- More possibilities on next slide



Ambiguity Problem

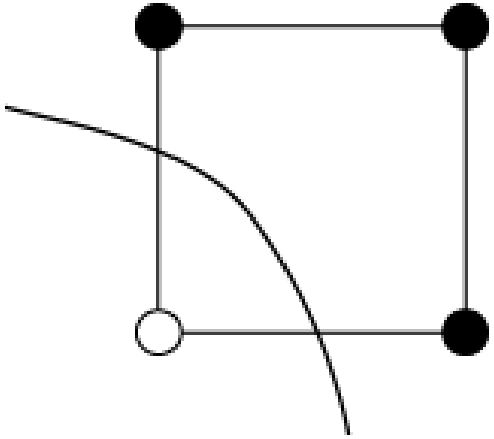


Is Problem Resolvable?

- 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

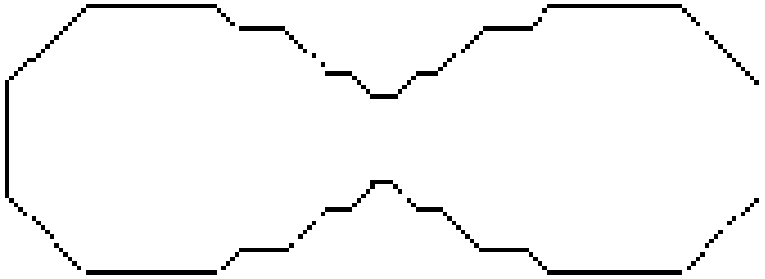


Example: Oval of Cassini

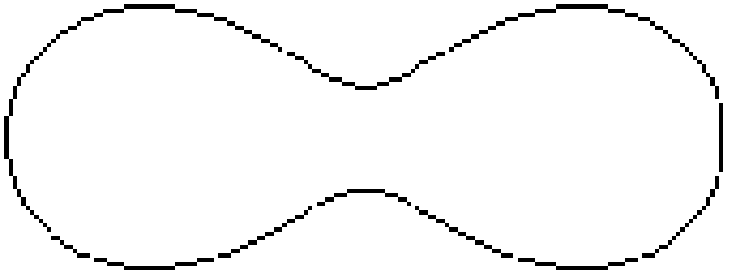
$$f(x,y)=(x^2+y^2+a^2)^2-4a^2x^2-b^4$$

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

midpoint intersections

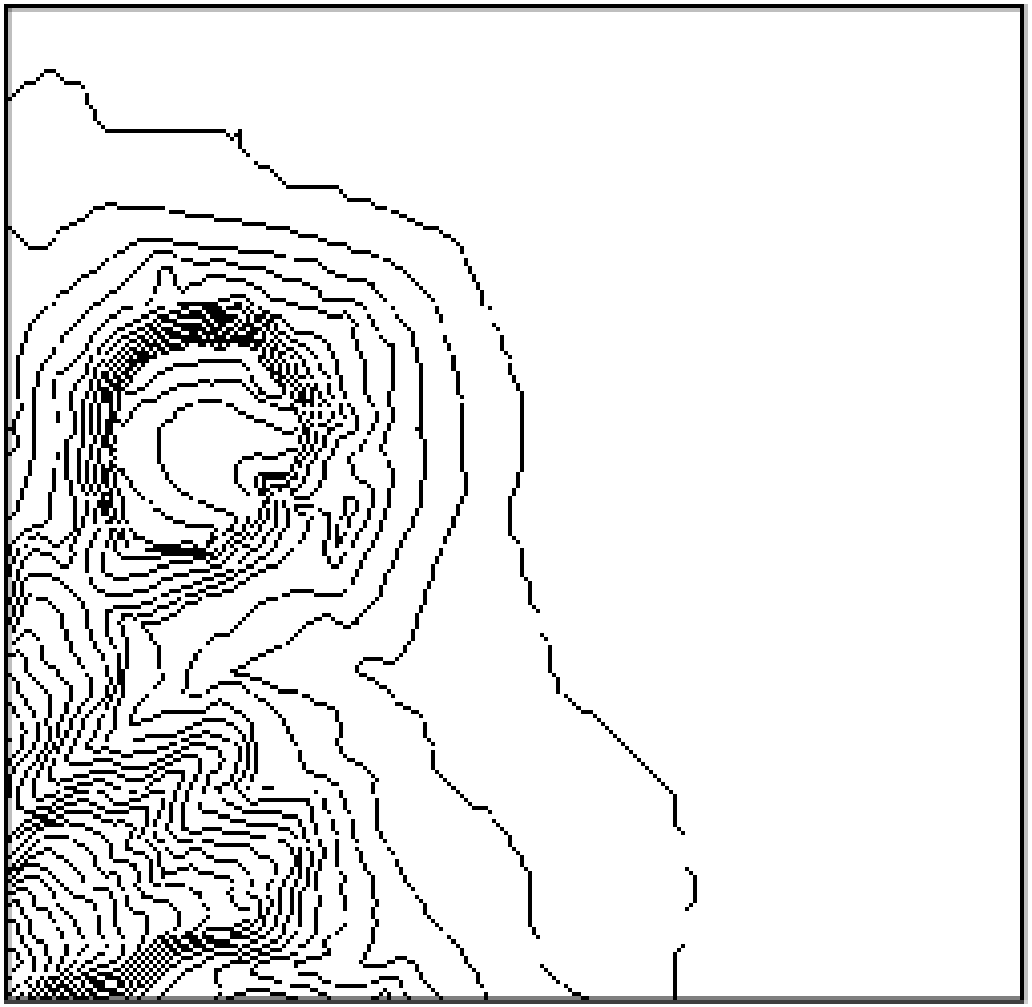


interpolating intersections



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