CS 247 – Scientific Visualization
Lecture 8: Scalar Fields, Pt. 4 [preview]

Markus Hadwiger, KAUST

Reading Assignment #4 (until Feb 21)

Read (required):

- Real-Time Volume Graphics book, Chapter 5 until 5.4 inclusive (*Terminology, Types of Light Sources, Gradient-Based Illumination, Local Illumination Models*)
- Paper:
  Marching Cubes: A high resolution 3D surface construction algorithm,
  Bill Lorensen and Harvey Cline, ACM SIGGRAPH 1987
  [> 17,700 citations and counting...]

  `https://dl.acm.org/doi/10.1145/37402.37422`

Read (optional):

- Paper:
  Flying Edges, William Schroeder et al., IEEE LDAV 2015

  `https://ieeexplore.ieee.org/document/7348069`

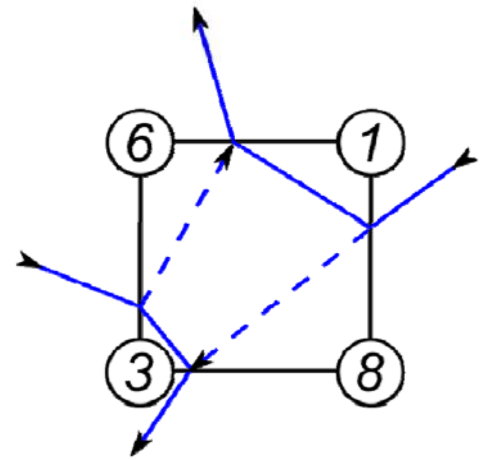
Scalar Fields

Ambiguities of contours

What is the correct contour of c=4?

Two possibilities, both are orientable:

- connect high values ————————
- connect low values - - - - - - - - - -



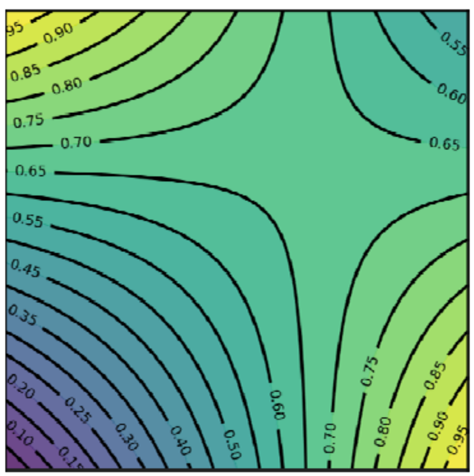
Answer: correctness depends on interior values of $f(x)$.

But: different interpolation schemes are possible.

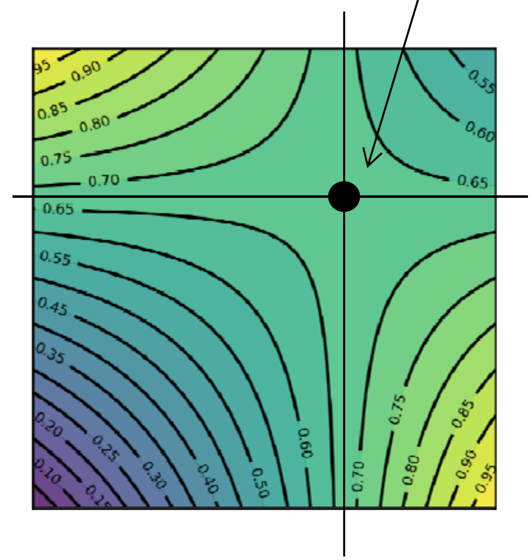
Better question: What is the correct contour with respect to bilinear interpolation?

Bi-Linear Interpolation: Critical Points

Critical points are where the gradient vanishes (i.e., is the zero vector)



critical point
(saddle point)



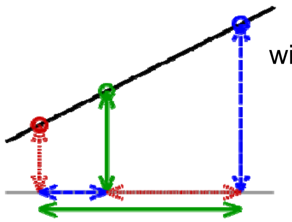
here, the critical
value is 2/3=0.666…

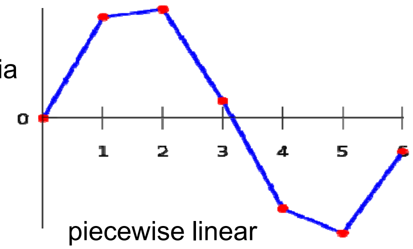
“Asymptotic decider”: resolve ambiguous configurations (6 and 9) by comparing specific iso-value with critical value (scalar value at critical point)

Linear Interpolation / Convex Combinations

Linear interpolation in 1D:

$$f(\alpha) = (1-\alpha)v_1 + \alpha v_2$$





Line embedded in 2D (linear interpolation of vertex coordinates/attributes):

$$f(\alpha_1, \alpha_2) = \alpha_1 v_1 + \alpha_2 v_2 \qquad f(\alpha) = v_1 + \alpha(v_2 - v_1)$$

$$\alpha_1 + \alpha_2 = 1 \qquad \alpha = \alpha_2$$

Line segment: $\alpha_1, \alpha_2 \geq 0$ (→ convex combination)

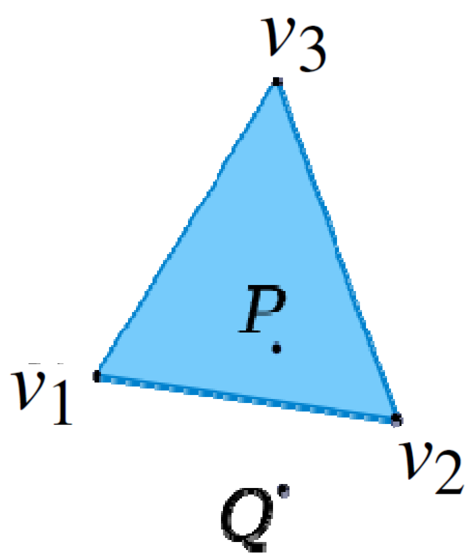
Compare to line parameterization with parameter t:

$$v(t) = v_1 + t(v_2 - v_1)$$

Linear Interpolation / Convex Combinations

Linear combination (n-dim. space):

$$\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n = \sum_{i=1}^{n} \alpha_i v_i$$



Affine combination: Restrict to $(n-1)$-dim. subspace:

$$\alpha_1 + \alpha_2 + \ldots + \alpha_n = \sum_{i=1}^{n} \alpha_i = 1$$

Convex combination: $\alpha_i \geq 0$

(restrict to simplex in subspace)

Linear Interpolation / Convex Combinations

The weights α_i are the n normalized ***barycentric*** coordinates

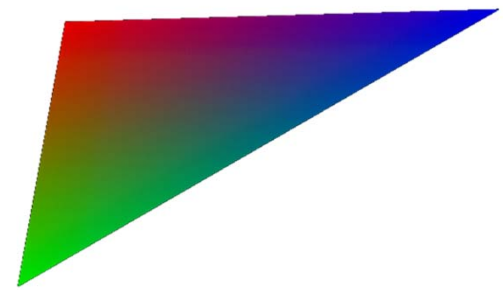
$\rightarrow$ linear attribute interpolation in simplex

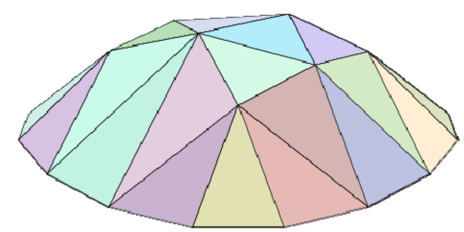
$$\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n = \sum_{i=1}^{n} \alpha_i v_i$$

$$\alpha_1 + \alpha_2 + \ldots + \alpha_n = \sum_{i=1}^{n} \alpha_i = 1$$

$$\alpha_i \geq 0$$

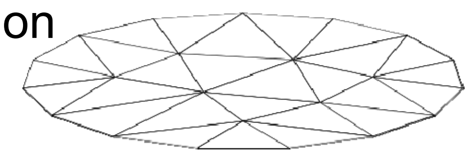
attribute interpolation





spatial position
interpolation

wikipedia



Linear Interpolation / Convex Combinations

$$\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n = \sum_{i=1}^{n} \alpha_i v_i$$

$$\alpha_1 + \alpha_2 + \ldots + \alpha_n = \sum_{i=1}^{n} \alpha_i = 1$$

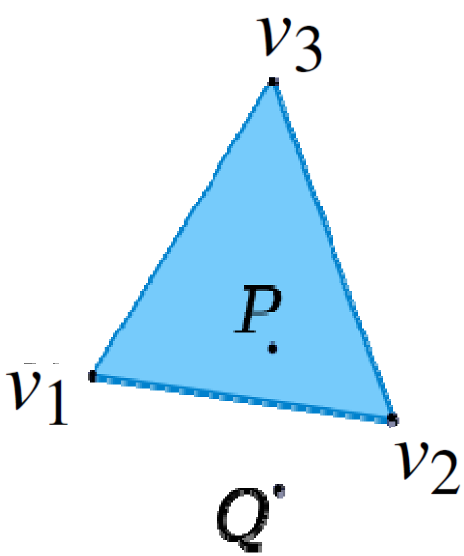
Can re-parameterize to get $(n-1)$ ***affine*** coordinates:

$$\alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 =$$

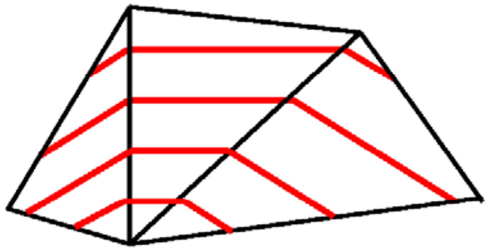
$$\tilde{\alpha}_1 (v_2 - v_1) + \tilde{\alpha}_2 (v_3 - v_1) + v_1$$

$$\tilde{\alpha}_1 = \alpha_2$$

$$\tilde{\alpha}_2 = \alpha_3$$



Contours in triangle/tetrahedral cells



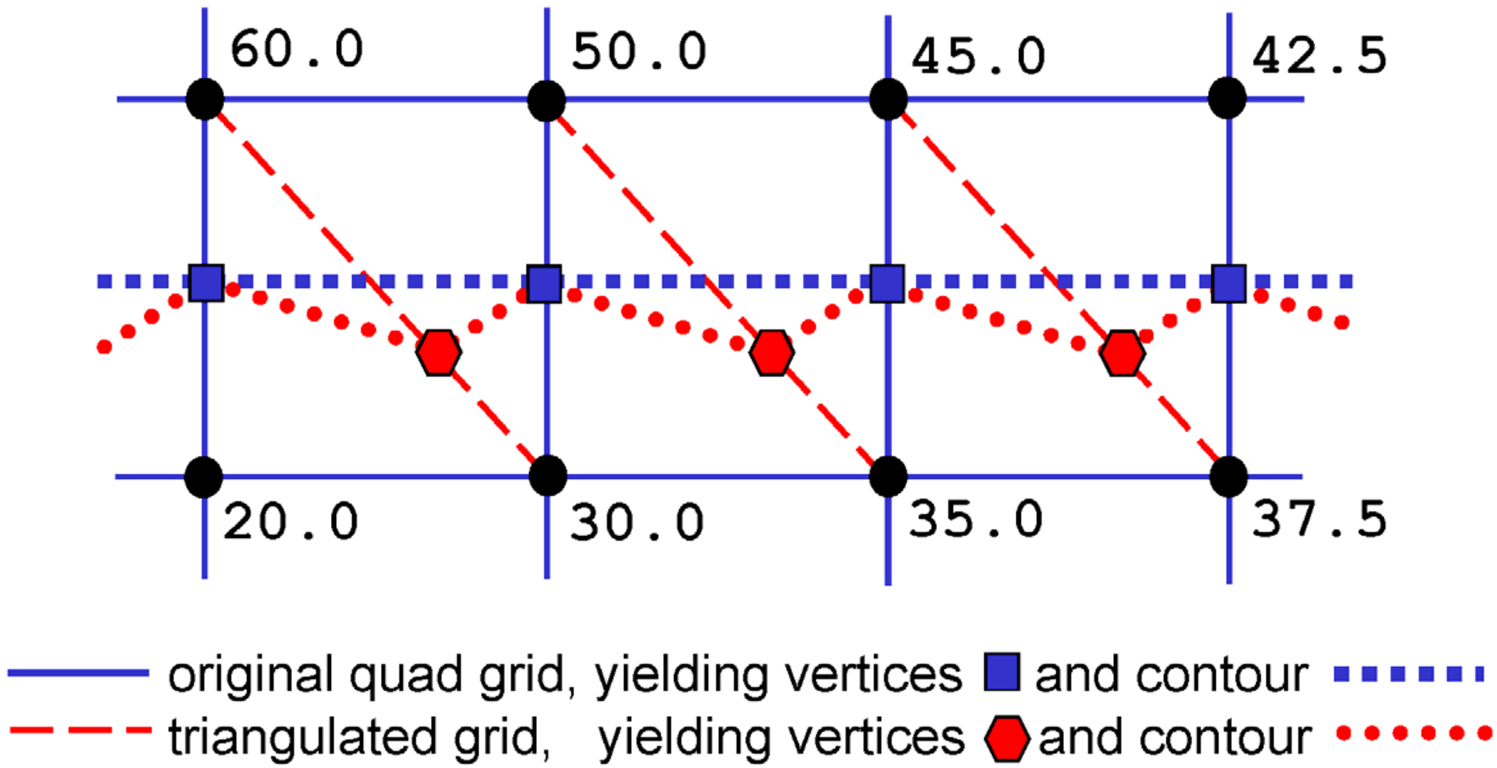
Linear interpolation of cells implies piece-wise linear contours.

Contours are unambiguous, making "marching triangles" even simpler than "marching squares".

Question: Why not split quadrangles into two triangles (and hexahedra into five or six tetrahedra) and use marching triangles (tetrahedra)?

Answer: This can introduce periodic artifacts!

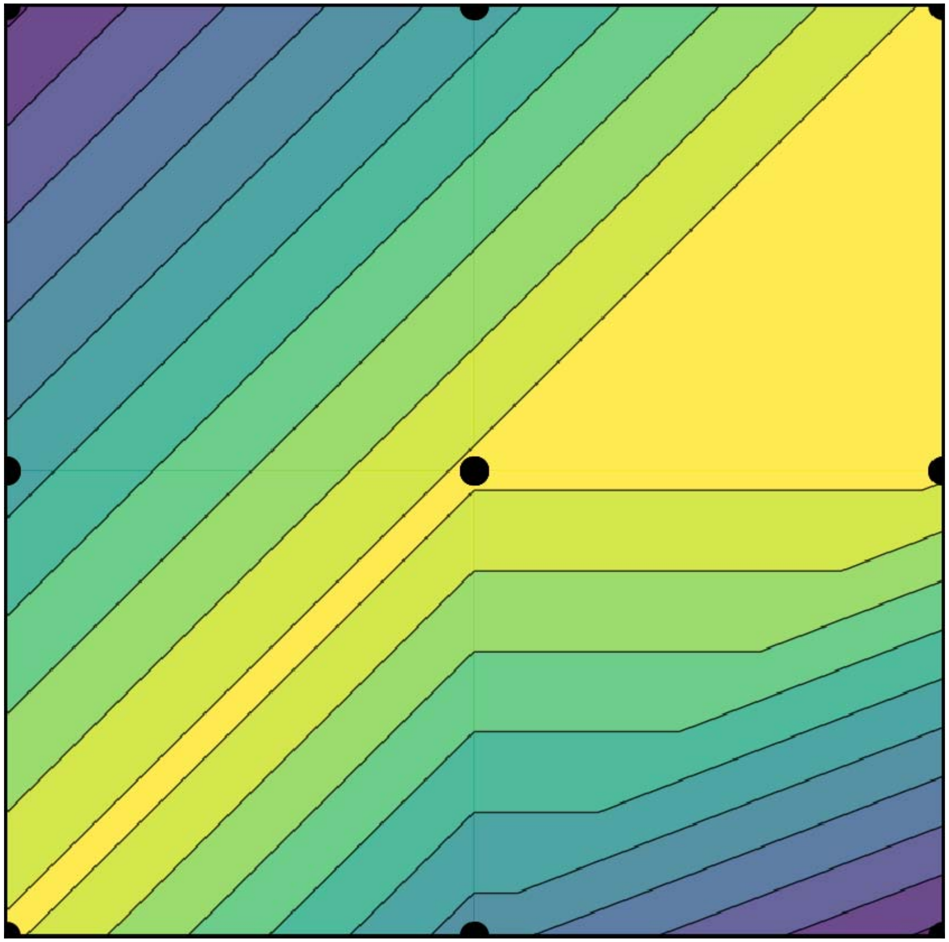
Illustrative example: Find contour at level *c*=40.0 !



Bi-Linear Interpolation: Comparisons

linear

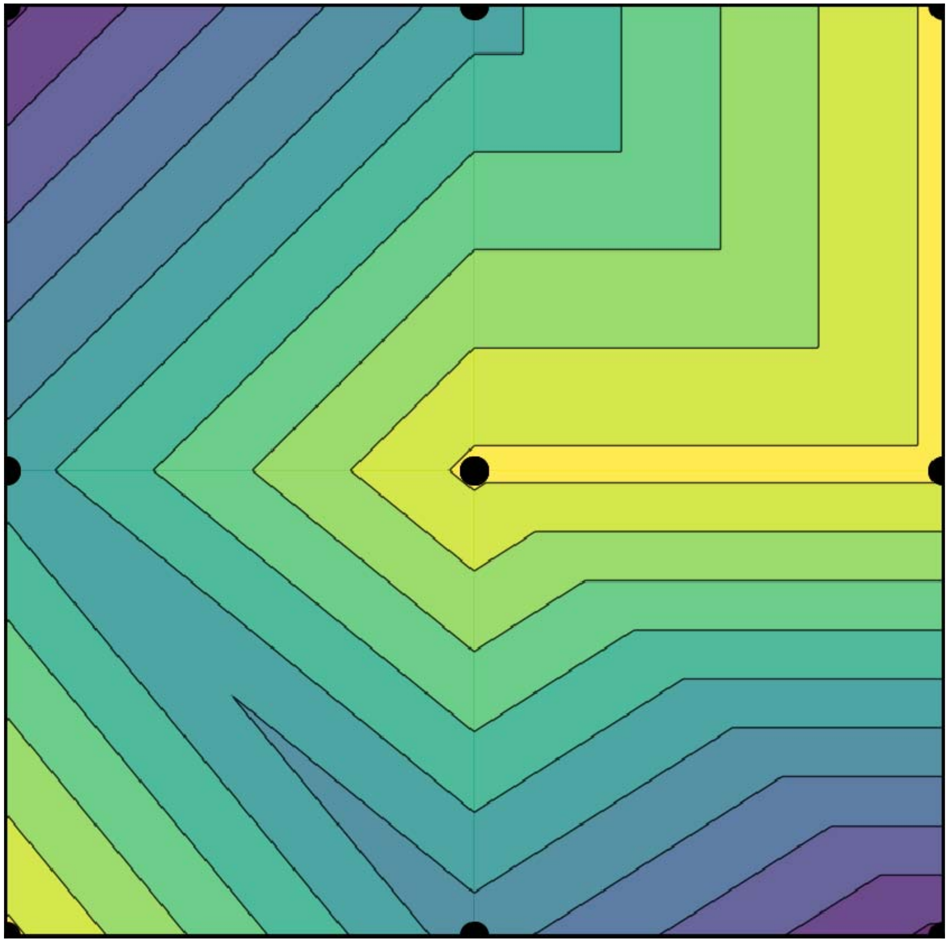
(2 triangles per quad;
diagonal:
bottom-left,
top-right)



Bi-Linear Interpolation: Comparisons

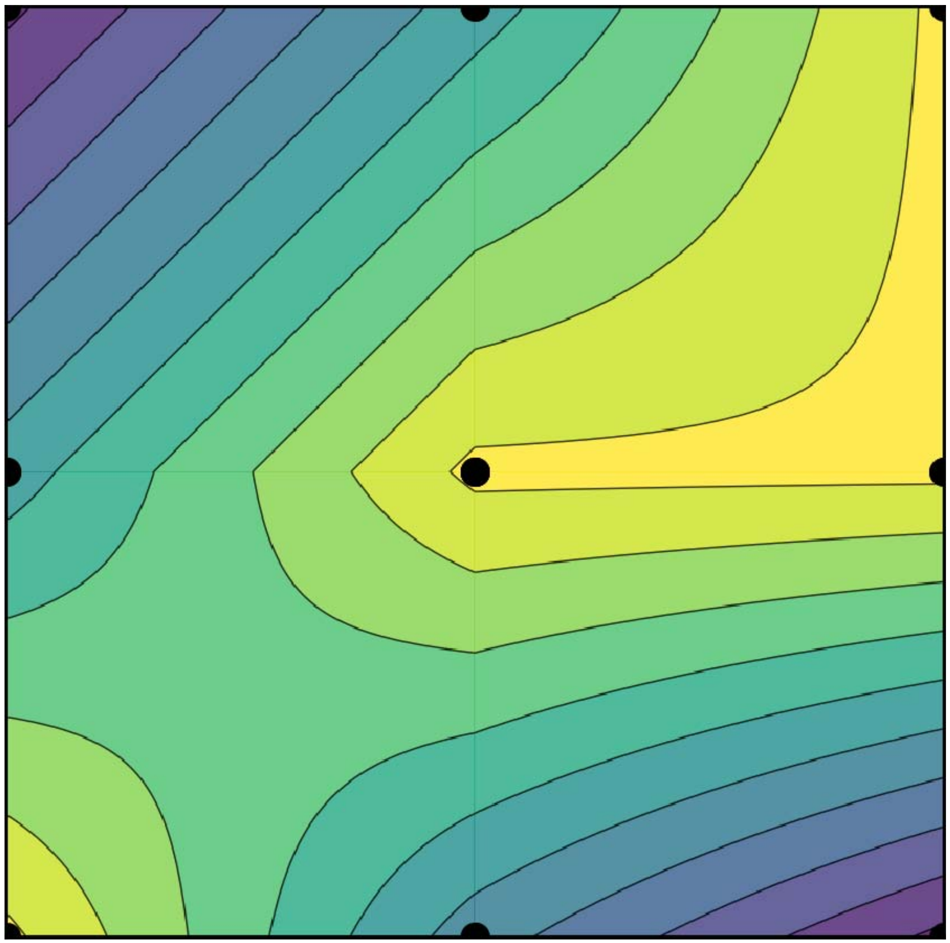
linear

(2 triangles per quad;
 diagonal:
 top-left,
 bottom-right)



Bi-Linear Interpolation: Comparisons

bi-linear



From 2D to 3D (Domain)

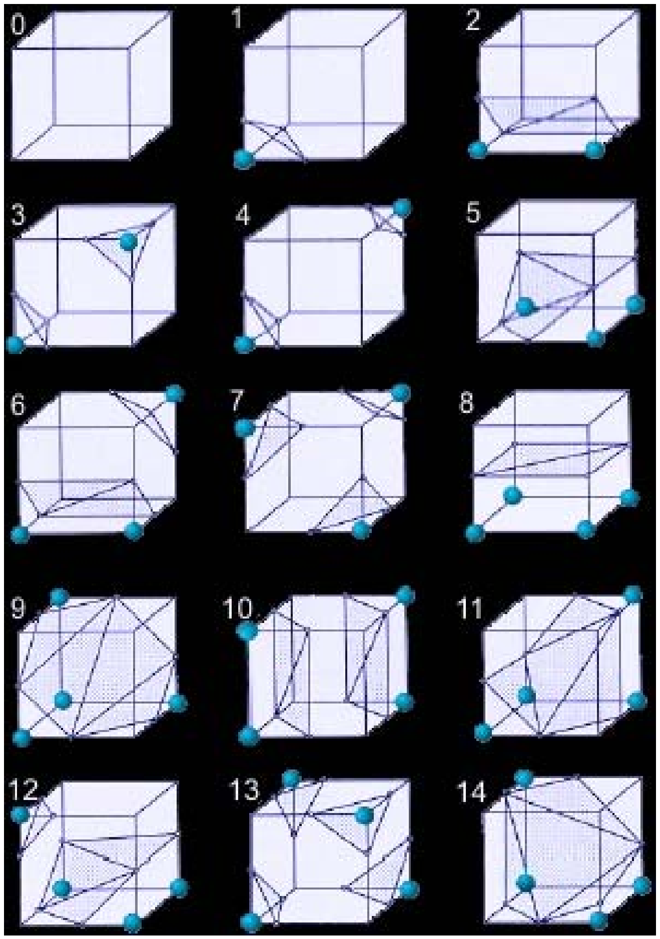
2D - Marching Squares Algorithm:

1. Locate the contour corresponding to a user-specified iso value
2. Create lines

3D - Marching Cubes Algorithm:

1. Locate the surface corresponding to a user-specified iso value
2. Create triangles
3. Calculate normals to the surface at each vertex
4. Draw shaded triangles

Marching Cubes



- For each cell, we have 8 vertices with 2 possible states each (inside or outside).
- This gives us 2^8 possible patterns = 256 cases.
- Enumerate cases to create a LUT
- Use symmetries to reduce problem from 256 to 15 cases.

Explanations

- Data Visualization book, 5.3.2
- Marching Cubes: A high resolution 3D surface construction algorithm, Lorensen & Cline, ACM SIGGRAPH 1987

The marching cubes algorithm

Contours of 3D scalar fields are known as isosurfaces.

Before 1987, isosurfaces were computed as

- contours on planar slices, followed by
- "contour stitching".

The marching cubes algorithm computes contours directly in 3D.

- Pieces of the isosurfaces are generated on a cell-by-cell basis.
- Similar to marching squares, a 8-bit number is computed from the 8 signs of $\tilde{f}(x_i)$ on the corners of a hexahedral cell.
- The isosurface piece is looked up in a table with 256 entries.

How to build up the table of 256 cases?

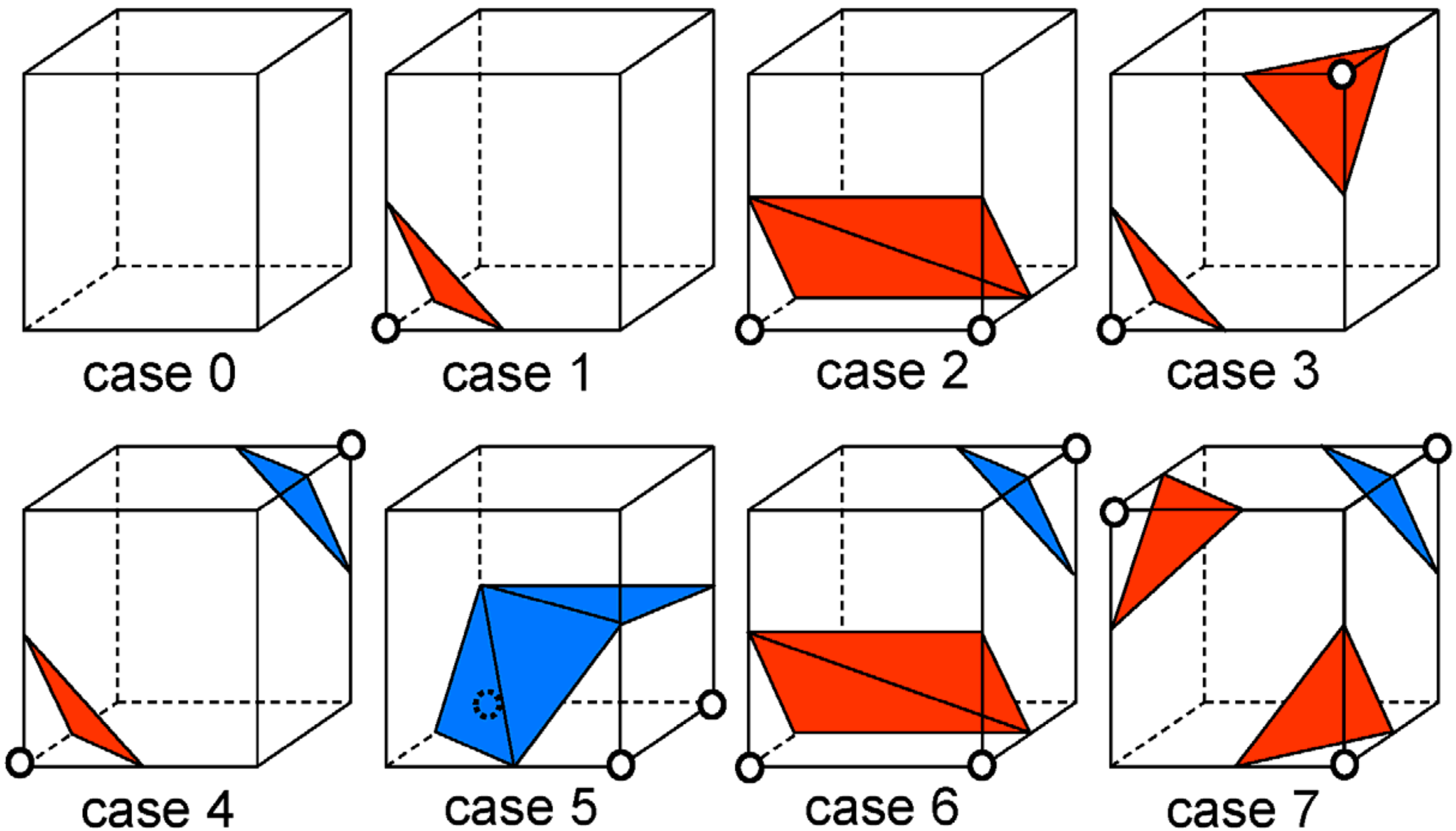
Lorensen and Cline (1987) exploited 3 types of symmetries:

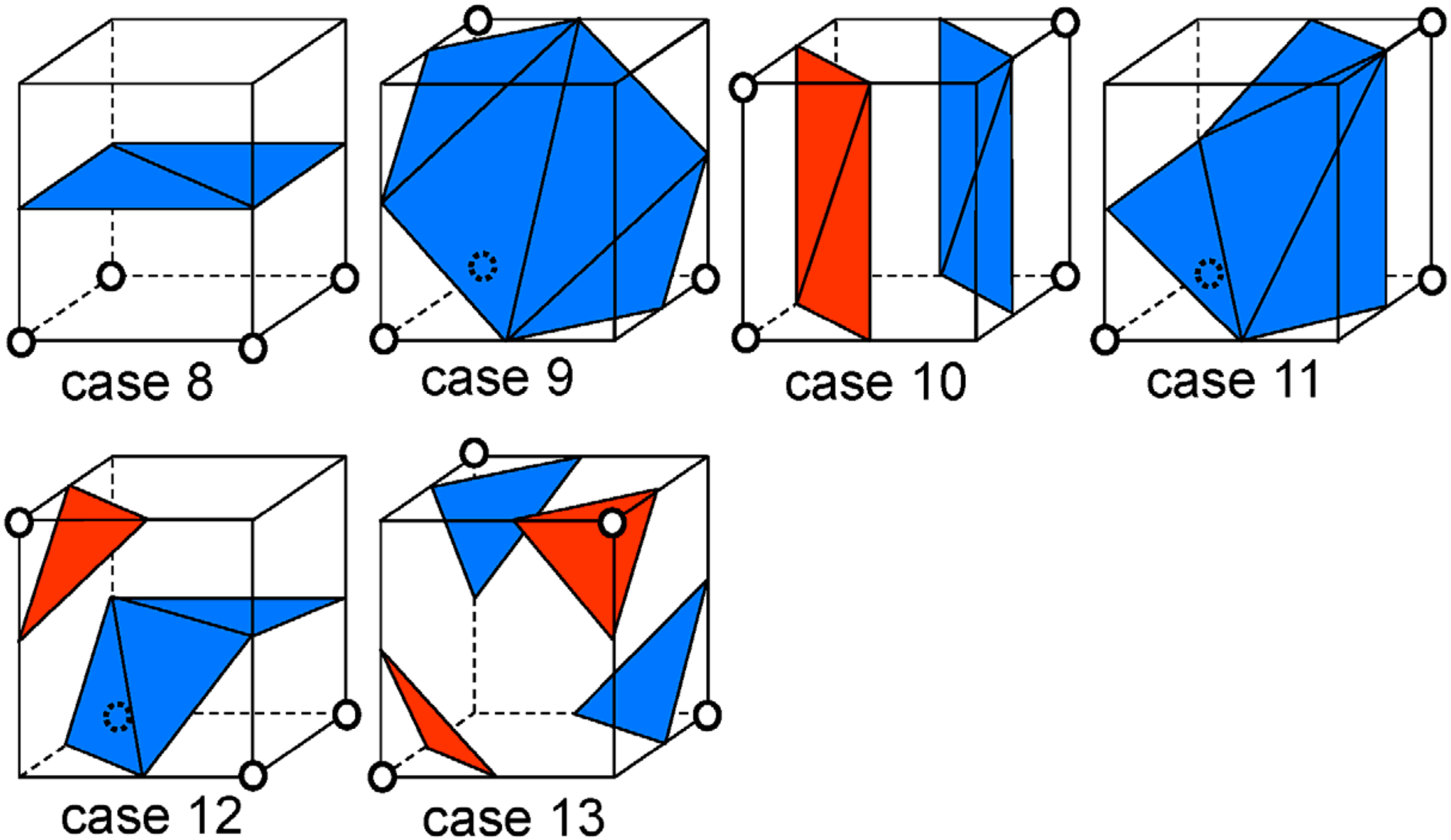
- rotational symmetries of the cube
- reflective symmetries of the cube
- sign changes of $\tilde{f}(x_i)$

They published a reduced set of 14*) cases shown on the next slides where

- white circles indicate positive signs of $\tilde{f}(x_i)$
- the positive side of the isosurface is drawn in red, the negative side in blue.

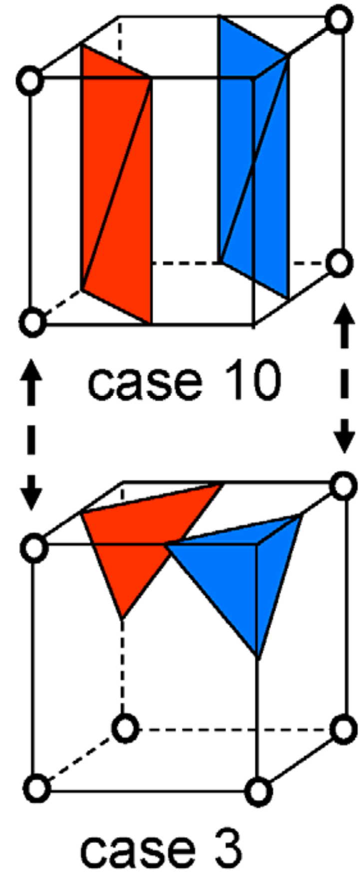
*) plus an unnecessary "case 14" which is a symmetric image of case 11.


case 0
case 1
case 2
case 3
case 4
case 5
case 6
case 7


case 8
case 9
case 10
case 11
case 12
case 13

Do the pieces fit together?

- The correct isosurfaces of the trilinear interpolant would fit (trilinear reduces to bilinear on the cell interfaces)
- but the marching cubes polygons don't necessarily fit.

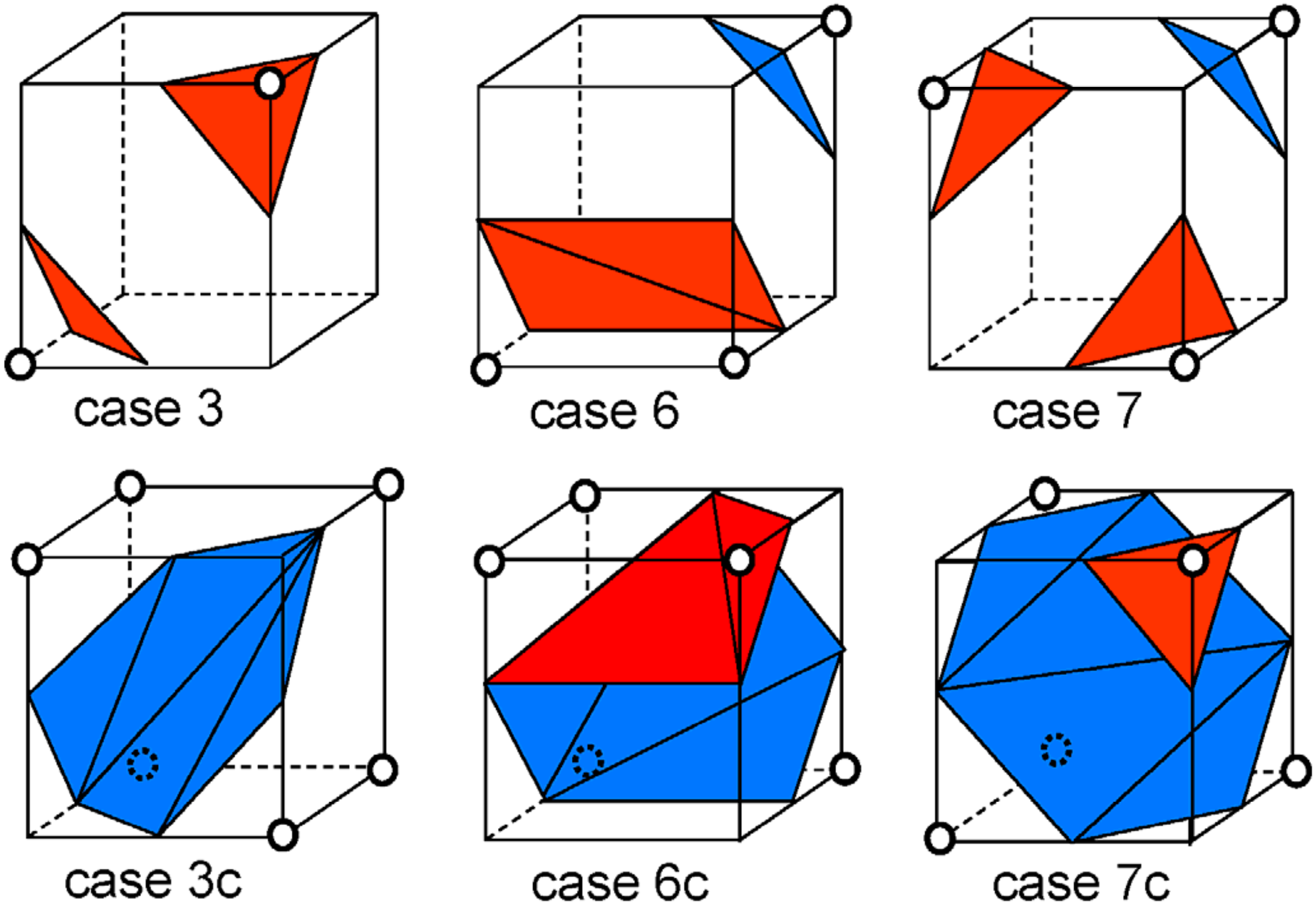


Example

- case 10, on top of
- case 3 (rotated, signs changed)

have matching signs at nodes but polygons don't fit.

The marching cubes algorithm



Summary of marching cubes algorithm:

Pre-processing steps:

- build a table of the 28 cases
- derive a table of the 256 cases, containing info on
  - intersected cell edges, e.g. for case 3/256 (see case 2/28): (0,2), (0,4), (1,3), (1,5)
  - triangles based on these points, e.g. for case 3/256: (0,2,1), (1,3,2).

Loop over cells:

- find sign of $\tilde{f}(x_i)$ for the 8 corner nodes, giving 8-bit integer
- use as index into (256 case) table
- find intersection points on edges listed in table, using linear interpolation
- generate triangles according to table

Post-processing steps:

- connect triangles (share vertices)
- compute normal vectors
  - by averaging triangle normals (problem: thin triangles!)
  - by estimating the gradient of the field $f(x_i)$ (better)

Triangle Mesh Data Structure (1)

Store list of vertices; vertices shared by triangles are replicated

Render, e.g., with OpenGL immediate mode, …

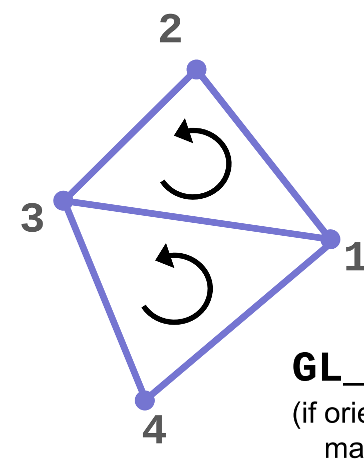
coords for vertex 1

```
x1,y1,z1
x2,y2,z2
x3,y3,z3
```
face 1

```
x1,y1,z1
x3,y3,z3
x4,y4,z4
```
face 2

```
...
```

```
struct face
  float verts[3][3]
  DataType val;
```



GL_CCW

(if orientable manifold)

Redundant, large storage size, cannot modify shared vertices easily

Store data values per face, or separately

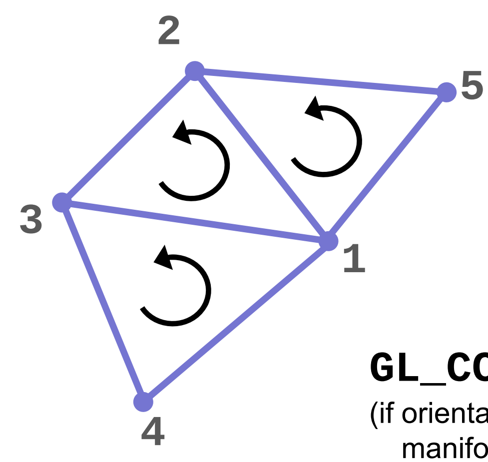
Triangle Mesh Data Structure (2)

Indexed face set: store list of vertices; store triangles as indexes

Render using separate vertex and index arrays / buffers

coords for vertex 1

vertex list	face list
`x1,y1,(z1)`	`1,2,3`
`x2,y2,(z2)`	`1,3,4`
`x3,y3,(z3)`	`2,1,5`
`x4,y4,(z4)`	`...`
`...`	



`GL_CCW`

(if orientable manifold)

Less redundancy, more efficient in terms of memory

Easy to change vertex positions; still have to do (global) search for shared edges (local information)

Orientability (2-manifold embedded in 3D)

not orientable

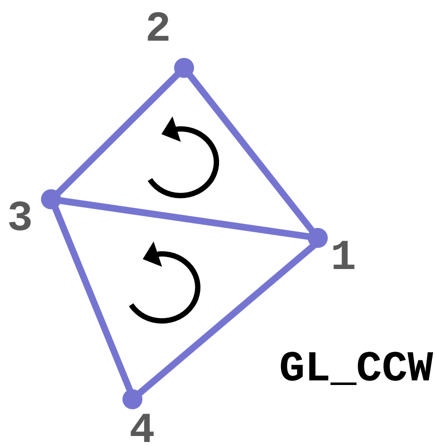
Moebius strip
(only one side!)

Orientability of 2-manifold:

Possible to assign consistent normal vector orientation

Triangle meshes

- Edges
  - Consistent ordering of vertices: CCW (counter-clockwise) or CW (clockwise)
    (e.g., (3,1,2) on one side of edge, (1,3,4) on the other side)
- Triangles
  - Consistent front side vs. back side
  - Normal vector; or ordering of vertices (CCW/CW)
  - See also: “right-hand rule”

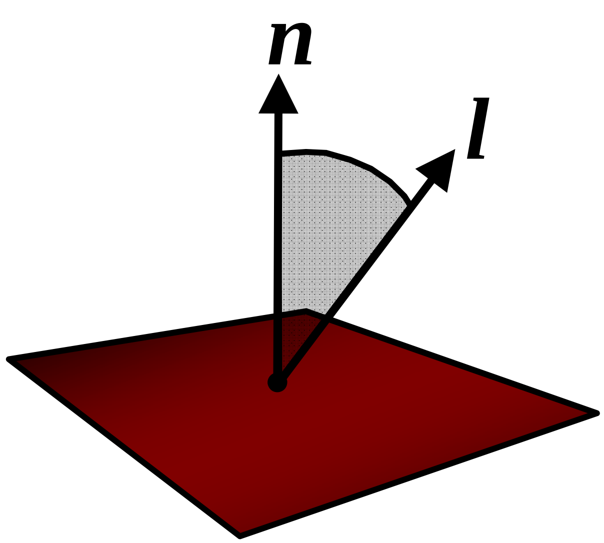


Local Shading Equations

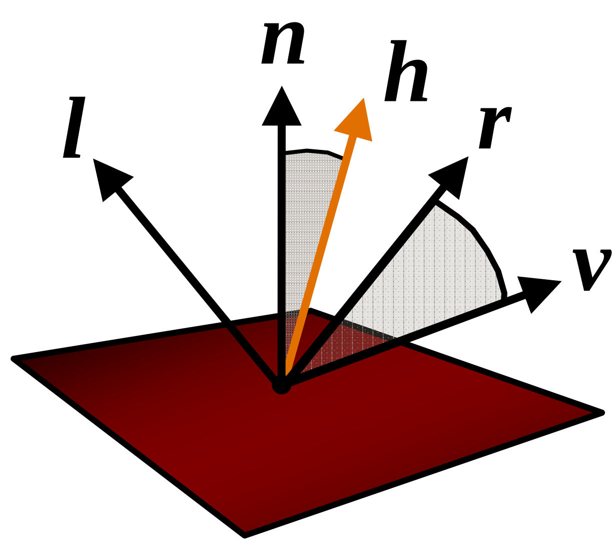
Standard volume shading adapts surface shading

Most commonly Blinn/Phong model

But what about the "surface" normal vector?



diffuse reflection

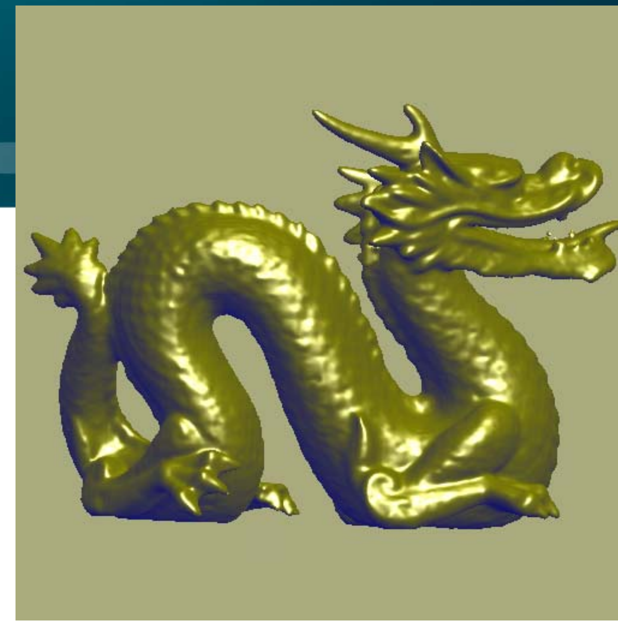


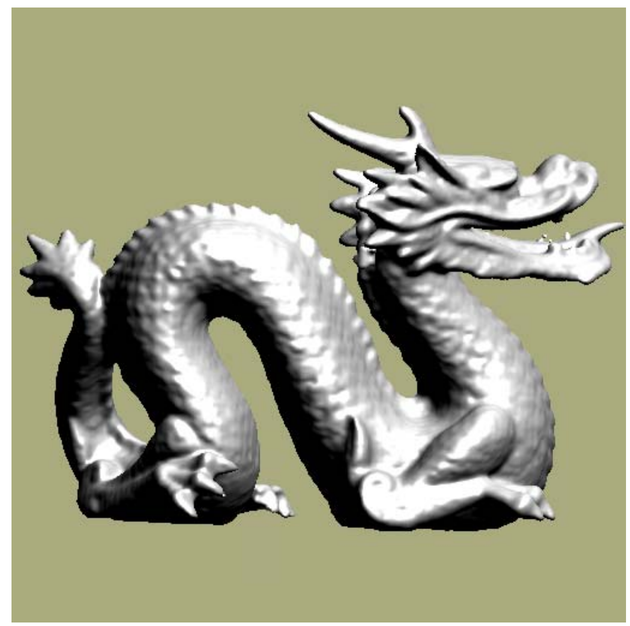
specular reflection

Iso-Surface / Volume Illumination

What About Volume Illumination?

Crucial for perceiving shape and depth relationships





this is a scalar volume (3D distance field)!

Local Illumination in Volumes

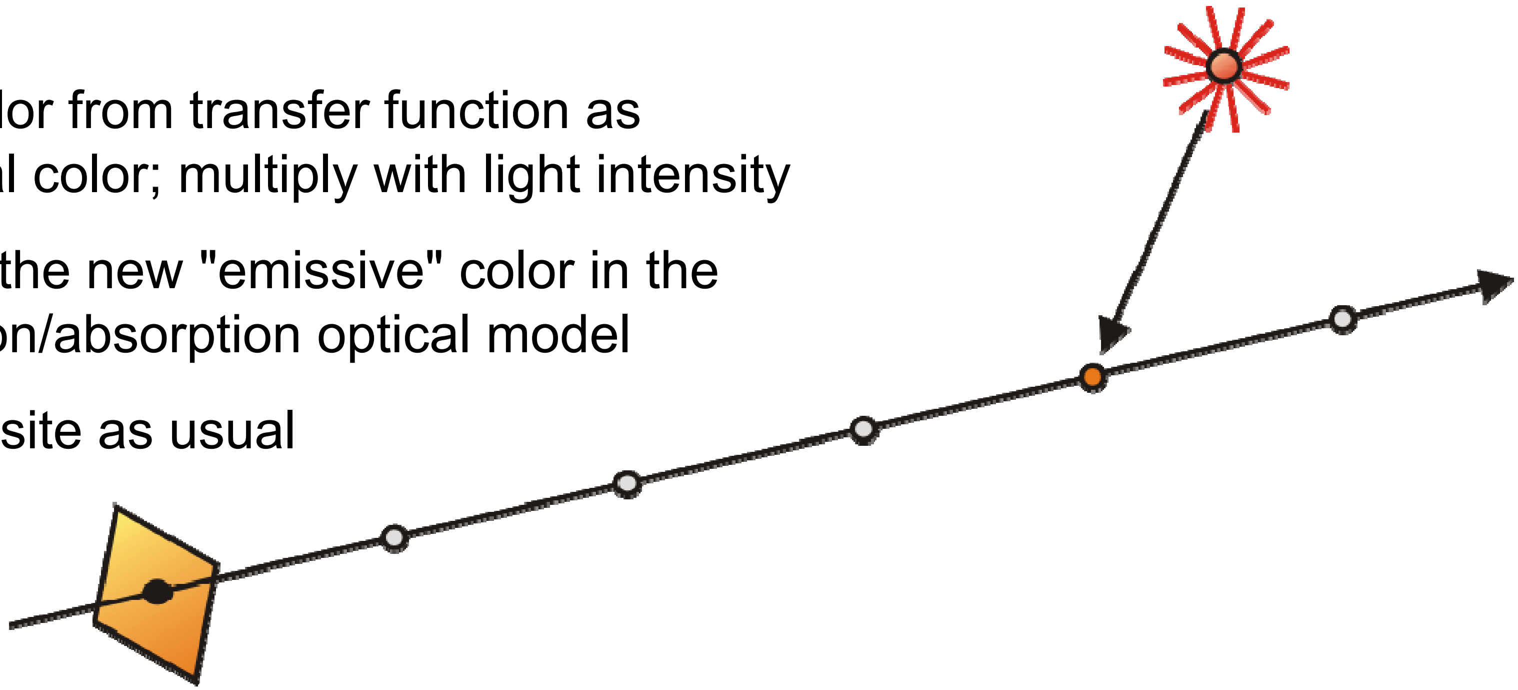
Interaction between light source and point in the volume

Local shading equation; evaluate at each point along a ray

Use color from transfer function as material color; multiply with light intensity

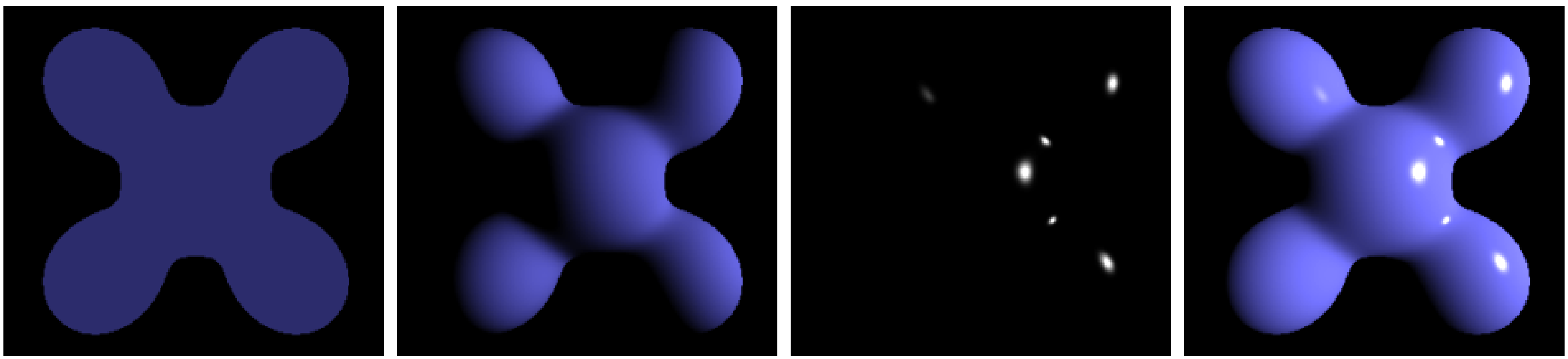
This is the new "emissive" color in the emission/absorption optical model

Composite as usual



Local Illumination Model: Phong Lighting Model

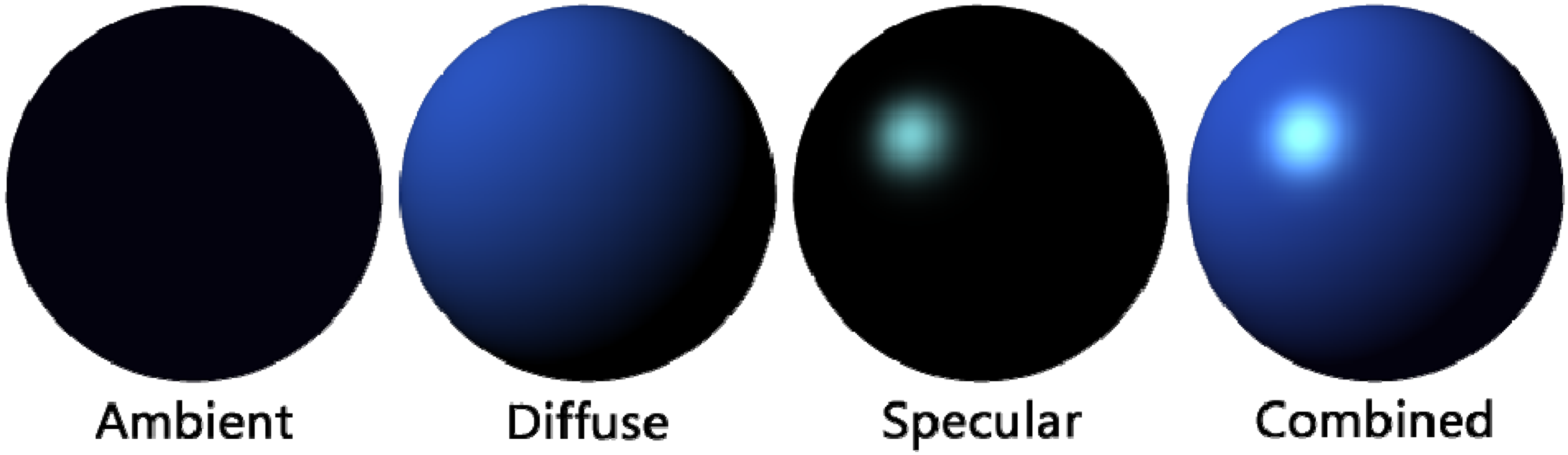
$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$



Ambient + Diffuse + Specular = Phong Reflection

Local Illumination Model: Phong Lighting Model

$$\mathbf{I}_{\mathrm{Phong}} = \mathbf{I}_{\mathrm{ambient}} + \mathbf{I}_{\mathrm{diffuse}} + \mathbf{I}_{\mathrm{specular}}$$

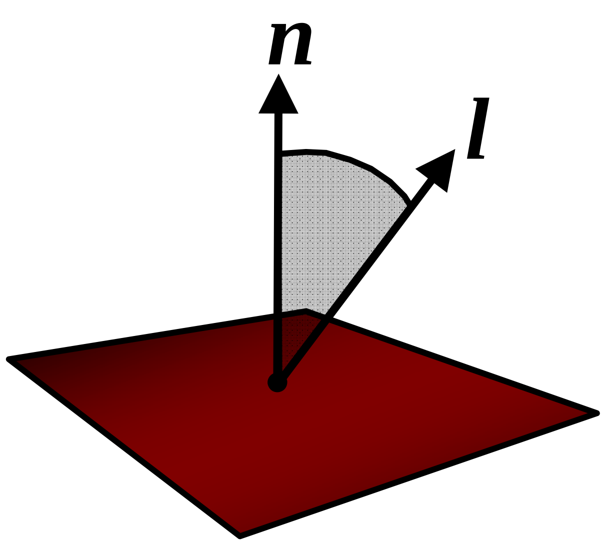


Local Shading Equations

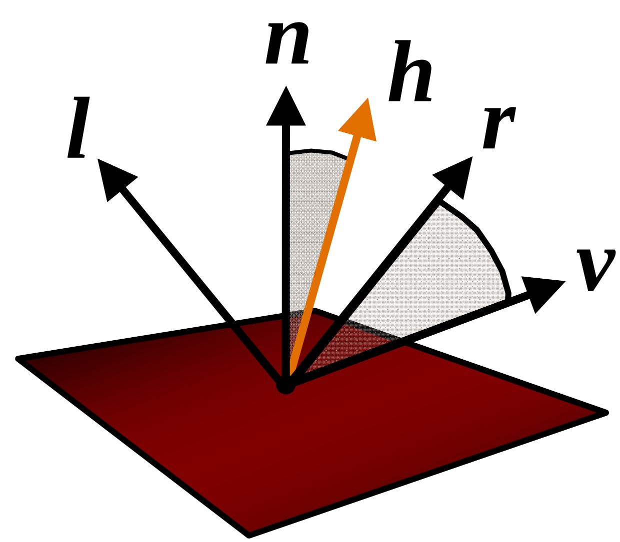
Standard volume shading adapts surface shading

Most commonly Blinn/Phong model

But what about the "surface" normal vector?



diffuse reflection



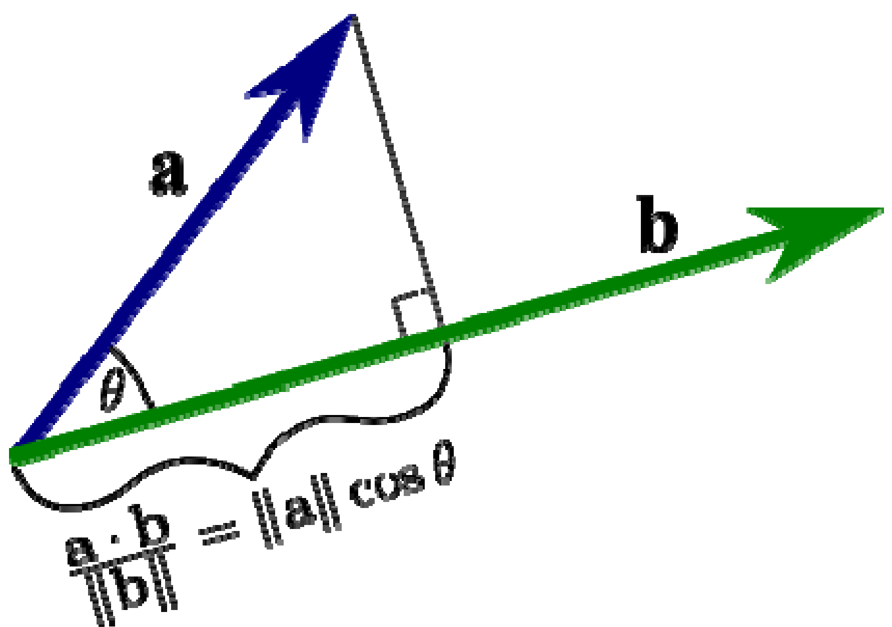
specular reflection

The Dot Product (Scalar / Inner Product)

Cosine of angle between two vectors times their lengths

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i \qquad \mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos\theta$$

(standard inner product in Cartesian coordinates)



Many uses:

- Project vector onto another vector,
  project into basis,
  project into tangent plane,
  ...

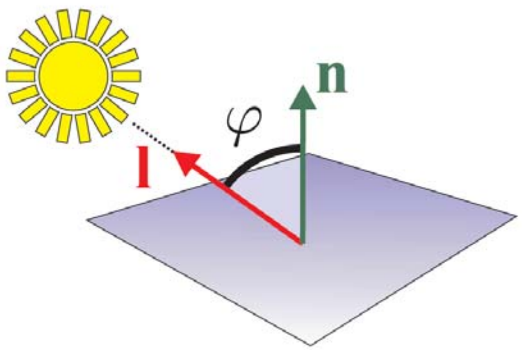
Local Illumination Model: Phong Lighting Model

$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$

$$\mathbf{I}_{\text{ambient}} = k_a \, \mathbf{M}_a \, \mathbf{I}_a$$

Local Illumination Model: Phong Lighting Model

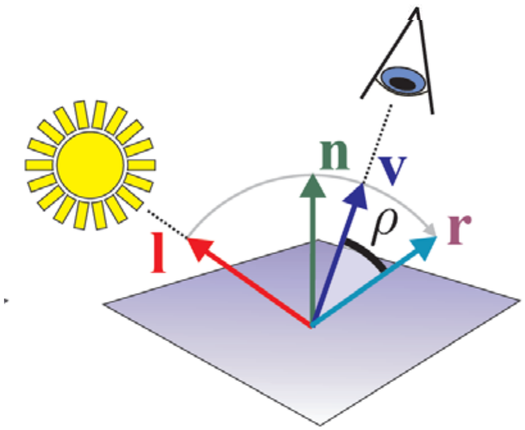
$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$



$$\begin{aligned}\mathbf{I}_{\text{diffuse}} &= k_d\,\mathbf{M}_d\,\mathbf{I}_d\cos\varphi \qquad \text{if } \varphi \leq \frac{\pi}{2}\\ &= k_d\,\mathbf{M}_d\,\mathbf{I}_d\max((\mathbf{n}\cdot\mathbf{l}),0)\end{aligned}$$

Local Illumination Model: Phong Lighting Model

$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$

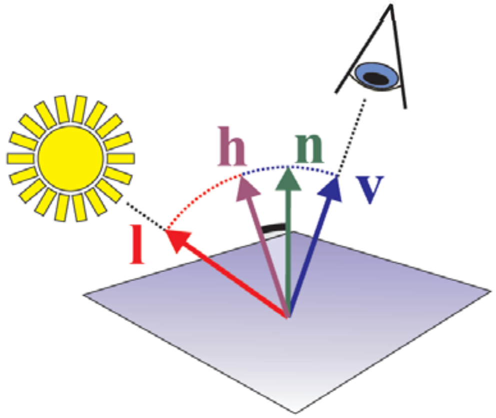


$$\begin{aligned} \mathbf{I}_{\text{specular}} &= k_s\, \mathbf{M}_s\, \mathbf{I}_s \cos^n \rho\,, \quad \text{if } \rho \leq \frac{\pi}{2} \\ &= k_s\, \mathbf{M}_s\, \mathbf{I}_s\, (\mathbf{r} \cdot \mathbf{v})^n \end{aligned}$$

must also clamp!

Local Illumination Model: Phong Lighting Model

$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$



$$\mathbf{I}_{\text{specular}} \approx k_s\,\mathbf{M}_s\,\mathbf{I}_s\,(\mathbf{h}\cdot\mathbf{n})^n$$

must also clamp!

$$\mathbf{h} = \frac{\mathbf{v}+\mathbf{l}}{\|\mathbf{v}+\mathbf{l}\|}$$

half-way vector

Gradient and Directional Derivative

Gradient $\nabla f(x,y,z)$ of scalar function $f(x,y,z)$: (in Cartesian coordinates)

$$\nabla f(x,y,z) = \left(\frac{\partial f(x,y,z)}{\partial x}, \frac{\partial f(x,y,z)}{\partial y}, \frac{\partial f(x,y,z)}{\partial z} \right)^T$$

Directional derivative in direction $\mathbf{u}$:

$$D_{\mathbf{u}} f(x,y,z) = \nabla f(x,y,z) \cdot \mathbf{u}$$

And therefore also:

$$D_{\mathbf{u}} f(x,y,z) = ||\nabla f|| \; ||\mathbf{u}|| \; \cos\theta$$

The Gradient as Normal Vector

Gradient of the scalar field gives direction+magnitude of fastest change

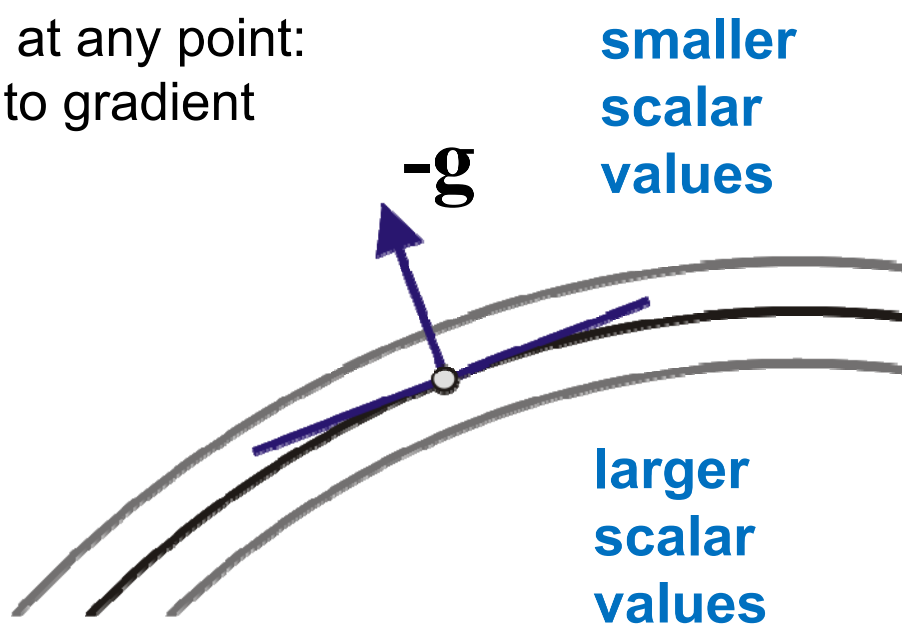
$$\mathbf{g} = \nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)^{\mathbf{T}}$$

(only correct in Cartesian coordinates [see later lectures])

Local approximation to isosurface at any point:
tangent plane = plane orthogonal to gradient

Normal of this isosurface:
normalized gradient vector
(negation is common convention)

$$\mathbf{n} = -\mathbf{g}/|\mathbf{g}|$$



Thank you.

Thanks for material

- Helwig Hauser
- Eduard Gröller
- Daniel Weiskopf
- Torsten Möller
- Ronny Peikert
- Philipp Muigg
- Christof Rezk-Salama