King Abdullah University of Science and Technology KAUST

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

Markus Hadwiger, KAUST

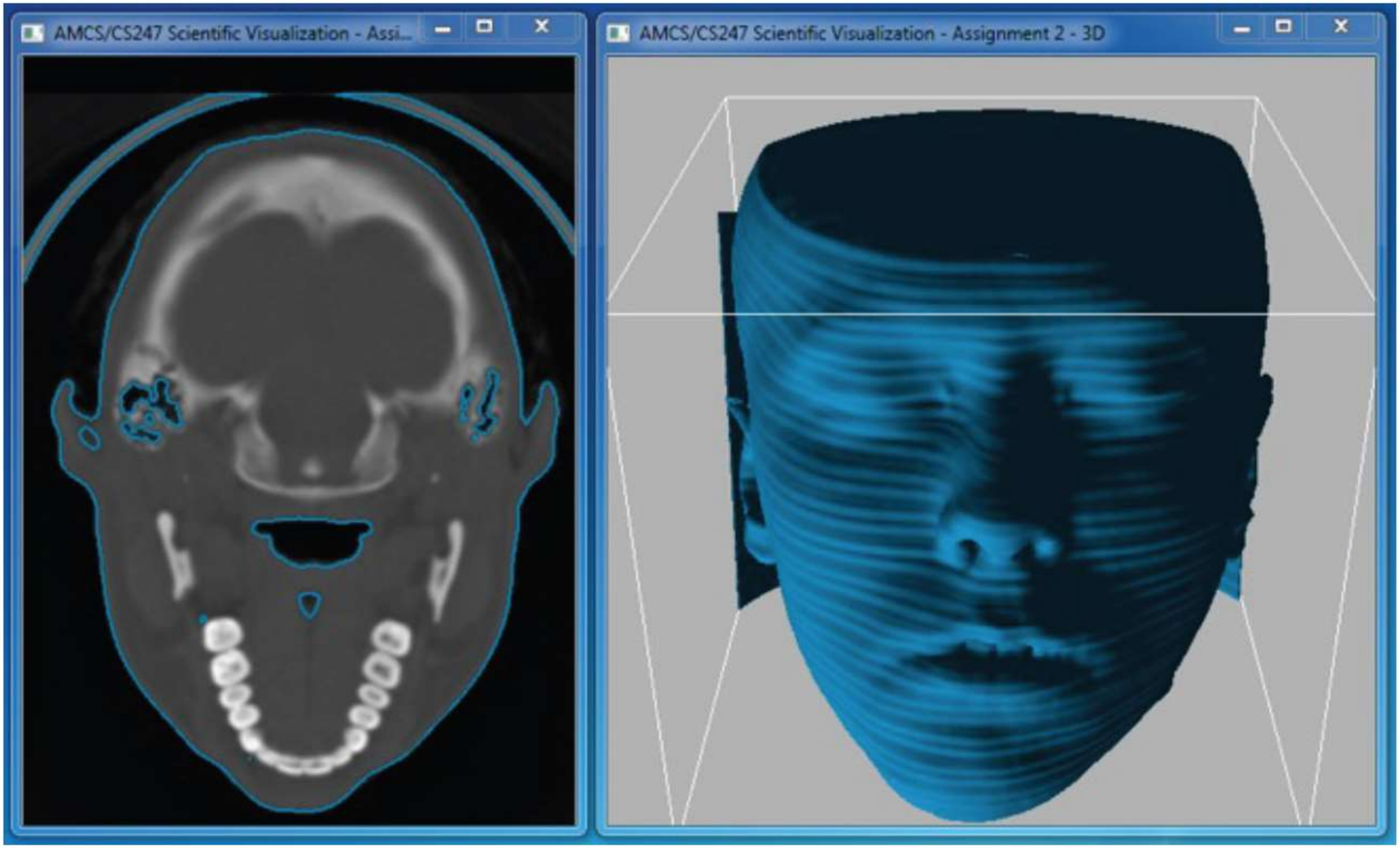
Reading Assignment #4 (until Feb 22)

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 & Harvey Cline, ACM SIGGRAPH 1987
  [> 16,000 citations and counting...]

  `http://dl.acm.org/citation.cfm?id=37422`

Programming Assignment 2 + 3



From 2D to 3D (Domain)

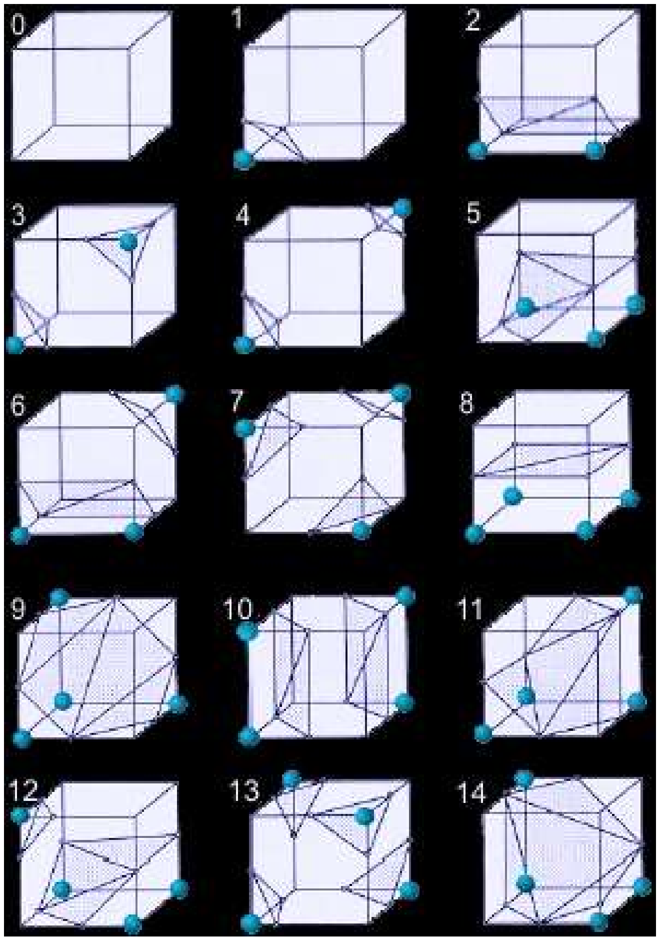
2D - Marching Squares Algorithm:

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:

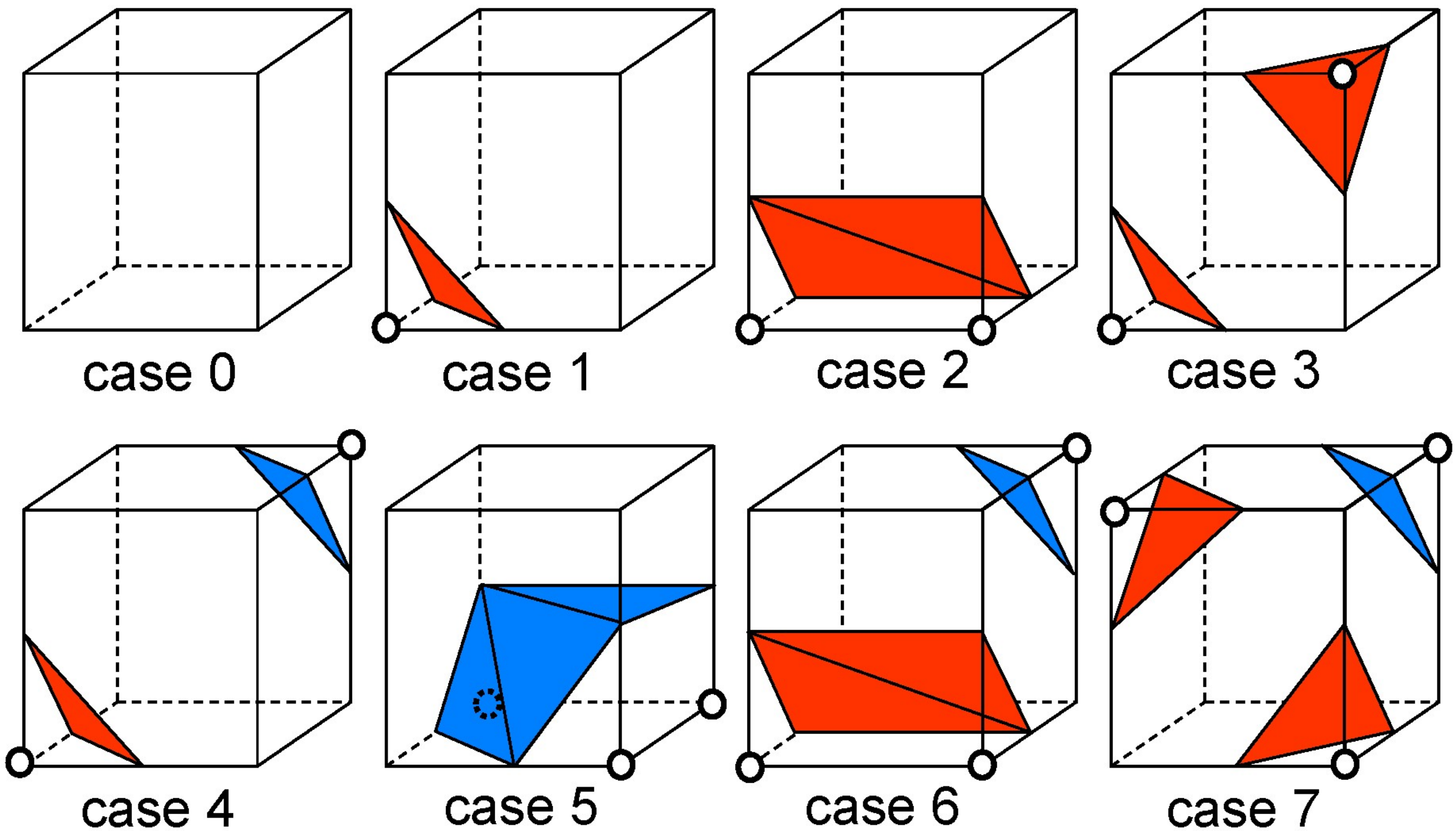
- rotational symmetries of the cube
- reflective symmetries of the cube
- sign changes of $\tilde{f}(x_i)$

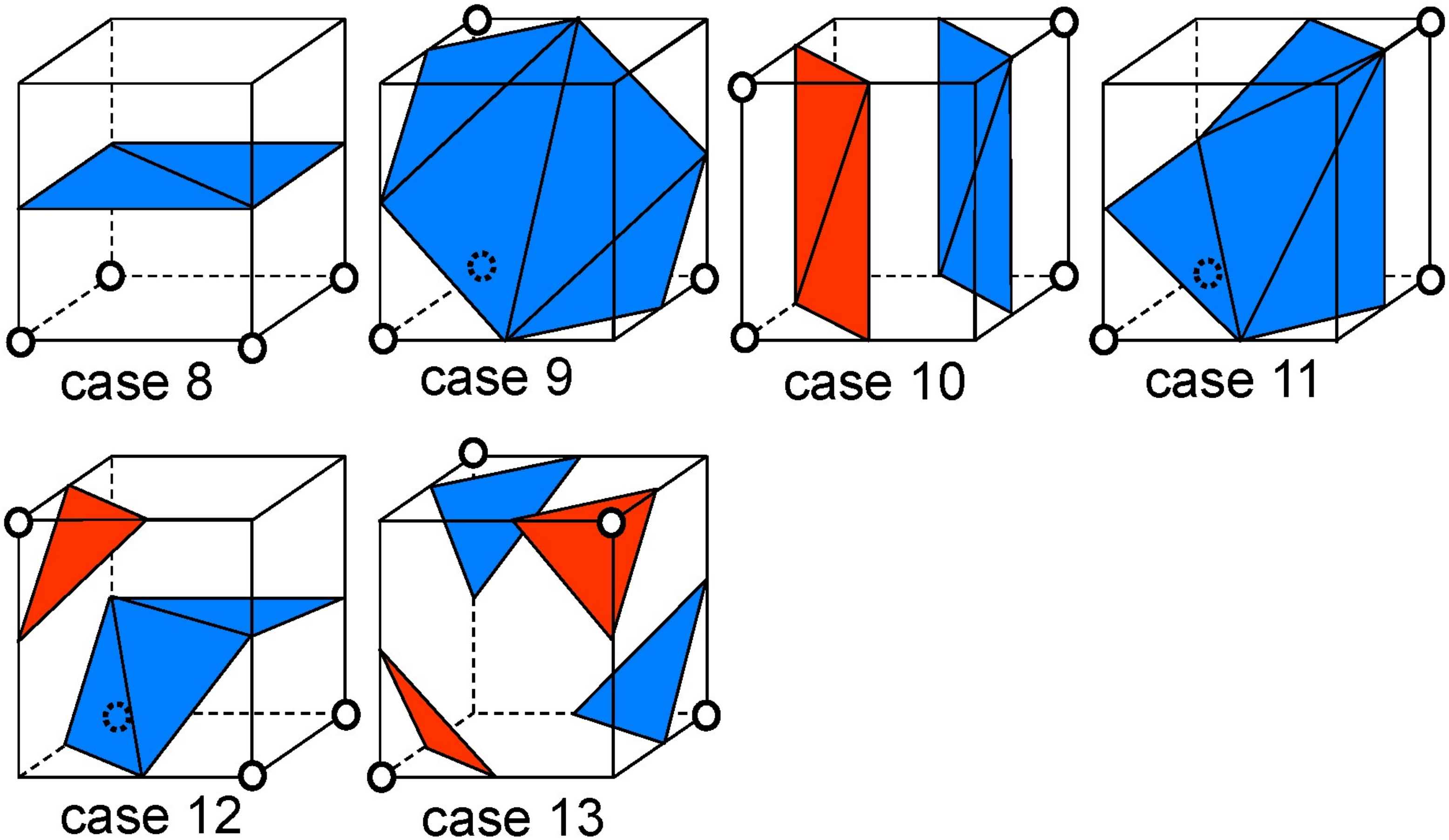
They published a reduced set of 14[*)] cases shown on the next slides where

- 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.

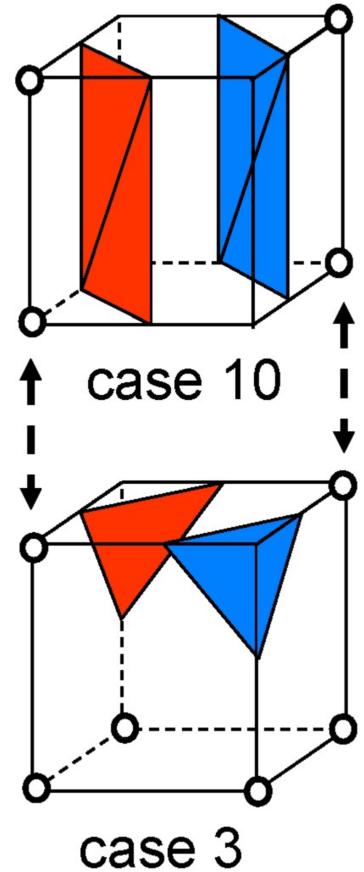
The marching cubes algorithm




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.

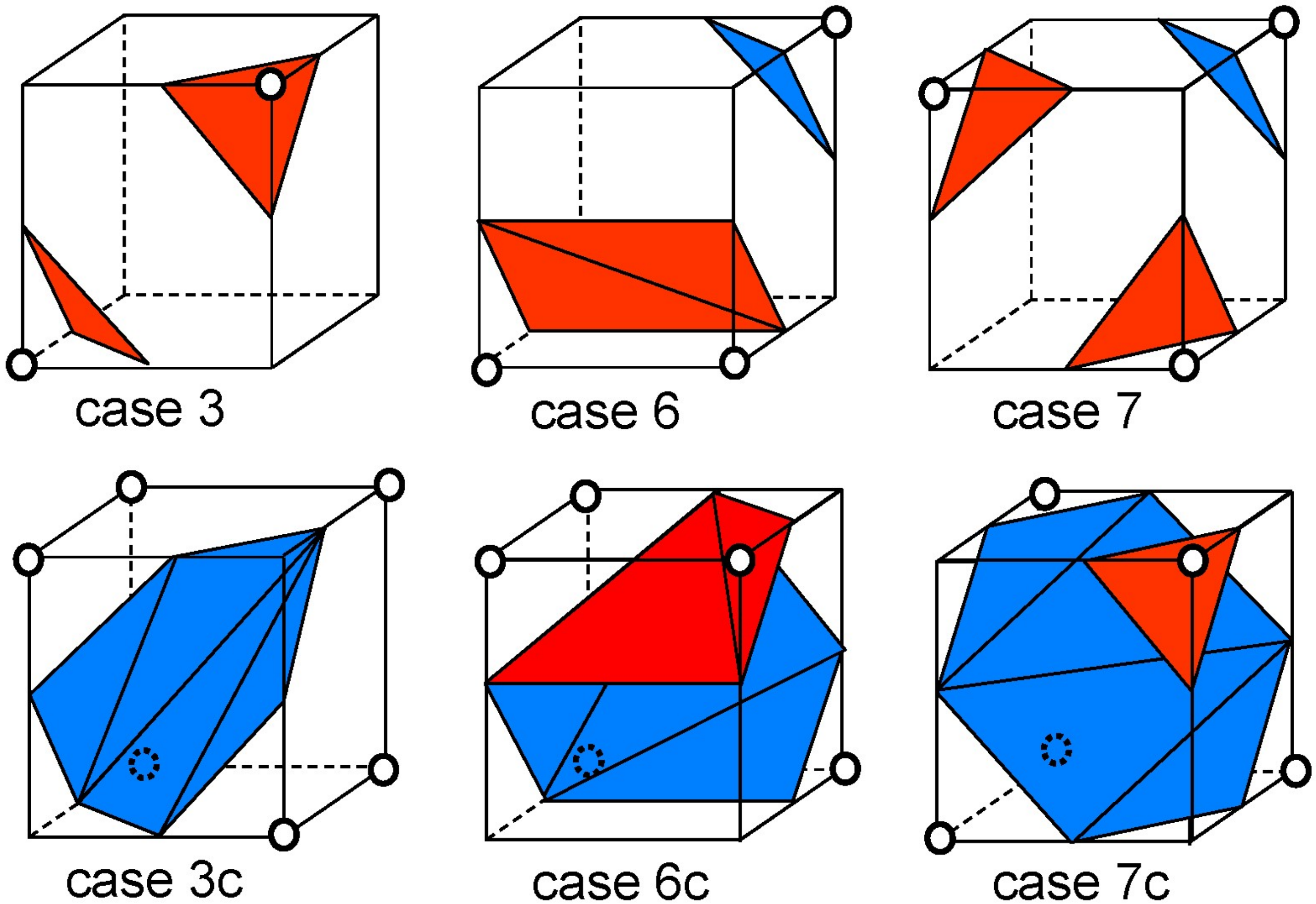


Example

- 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).

The marching cubes algorithm

Loop over cells:

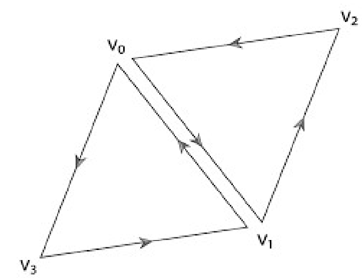
- find sign of $\tilde{f}(x_i)$ for the 8 corner nodes, giving 8-bit integer
- use as index into (256 case) table
- find intersection points on edges listed in table, using linear interpolation
- generate triangles according to table

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 Structures

- Typical implementations of unstructured grids
  - Direct form



coords for vertex 0

```
x0,y0,(z0)  ┐
x1,y1,(z1)  │ face 0
x2,y2,(z2)  ┘
x1,y1,(z1)  ┐
x0,y0,(z0)  │ face 1
x3,y3,(z3)  ┘

...
```

```
struct face
  float verts[3][2]          2D
  DataType val;

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

  - Additionally store the data values
  - Problems: storage space, redundancy, updates in multiple places

Triangle Mesh Data Structures

- Typical implementations of unstructured grids
  - Indirect form

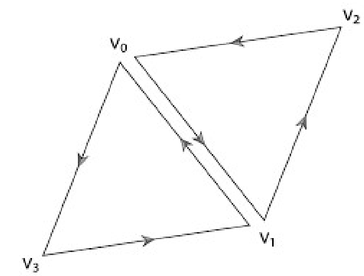
vertex list

coords for vertex 0

```
v[0]: x0,y0,z0
v[1]: x1,y1,z1
v[2]: x2,y2,z2
v[3]: x3,y3,z3
```

face list

```
f[0]: 0,1,2
f[1]: 1,0,3
f[2]: 2,1,4
...
```



  - Indexed face set
  - More efficient than direct approach in terms of memory requirements; geometry and topology separated
  - But still have to do global search to find local information (i.e. what faces share an edge)
  - More neighborhood information: half-edge data structure, …

Orientability (2-manifold embedded in 3D)

not orientable

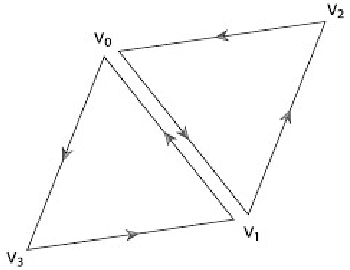
Moebius strip
(only one side!)

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., (0,1,2) on one side of edge, (1,0,3) on the other side)
- Triangles
  - Consistent front side vs. back side
  - Normal vector; or ordering of vertices (CCW/CW)
  - See also: “right-hand rule”

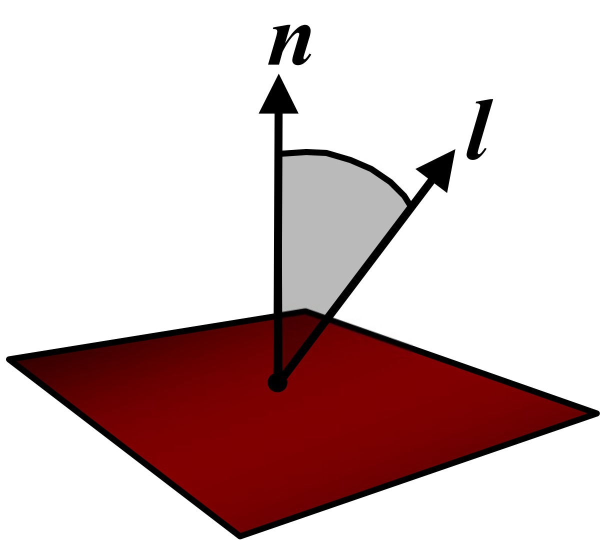


Local Shading Equations

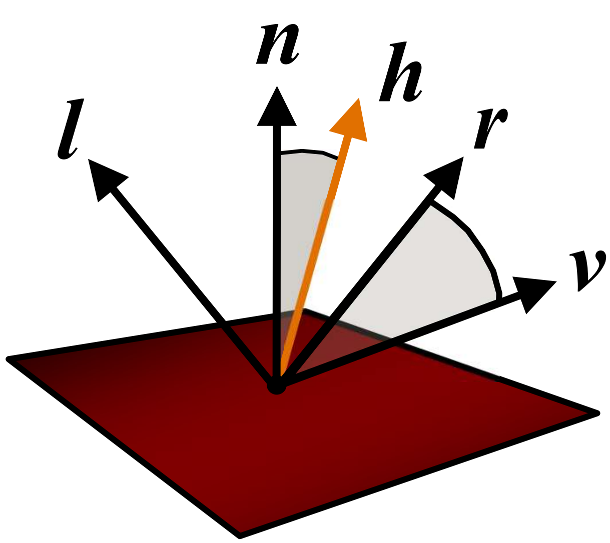
Standard volume shading adapts surface shading

Most commonly Blinn/Phong model

But what about the "surface" normal vector?



diffuse reflection



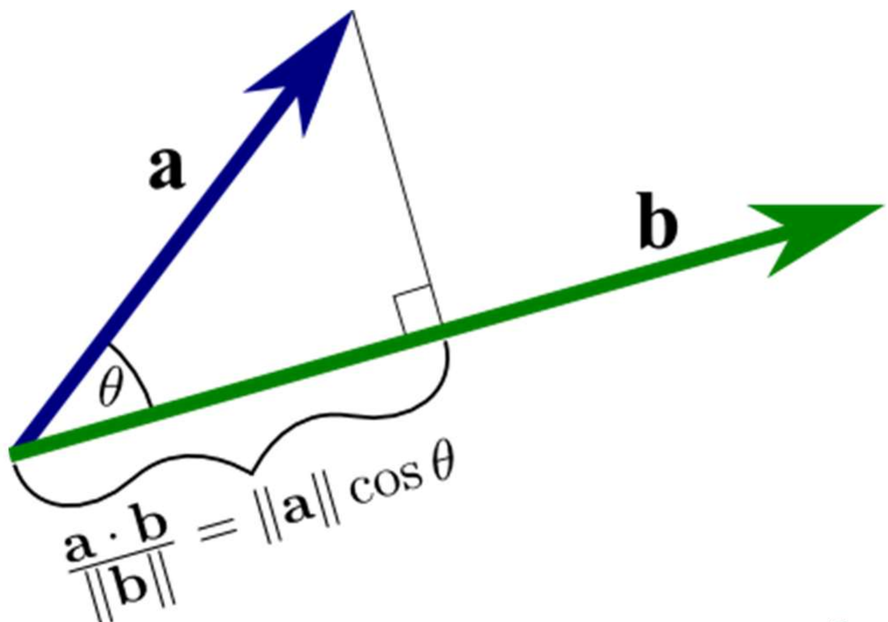
specular reflection

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,
  ...

Thank you.

Thanks for material

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