King Abdullah University of Science and Technology

KAUST

CS 247 – Scientific Visualization
Lecture 11: Scalar Field Visualization, Pt. 4

Markus Hadwiger, KAUST

Reading Assignment #6 (until Mar 9)

Read (required):

- Real-Time Volume Graphics, Chapter 2
  (*GPU Programming*)
- Reminder: Real-Time Volume Graphics, Chapter 5.4

Read (optional):

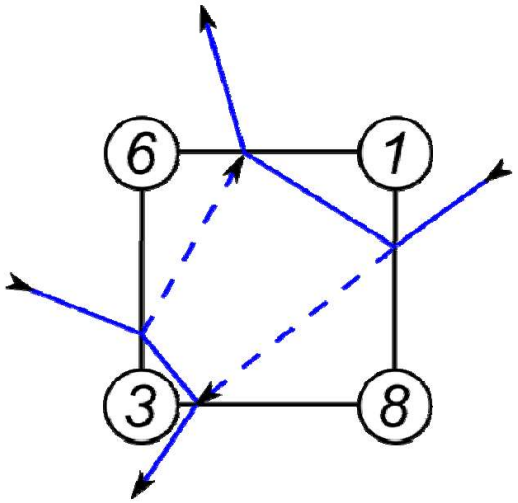
- Paper:
  Gregory M. Nielson and Bernd Hamann,
  The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes,
  Visualization 1991
  `https://dl.acm.org/doi/abs/10.5555/949607.949621`

Ambiguities of contours

What is the correct contour of c=4?

Two possibilities, both are orientable:

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



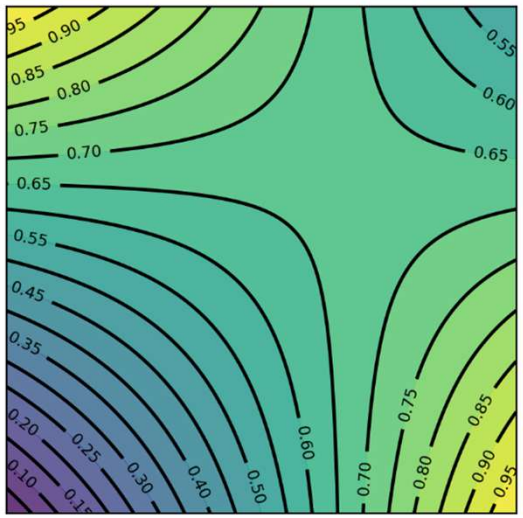
Answer: correctness depends on interior values of $f(x)$.

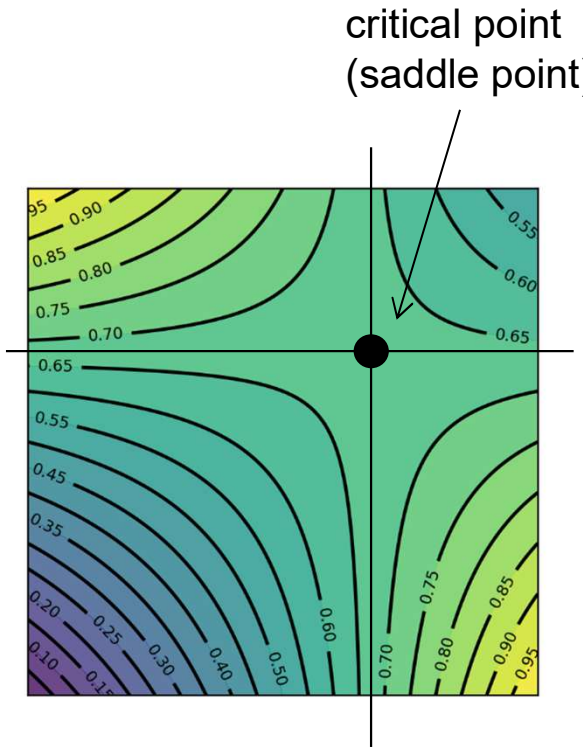
But: different interpolation schemes are possible.

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)





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)

From 2D to 3D (Domain)

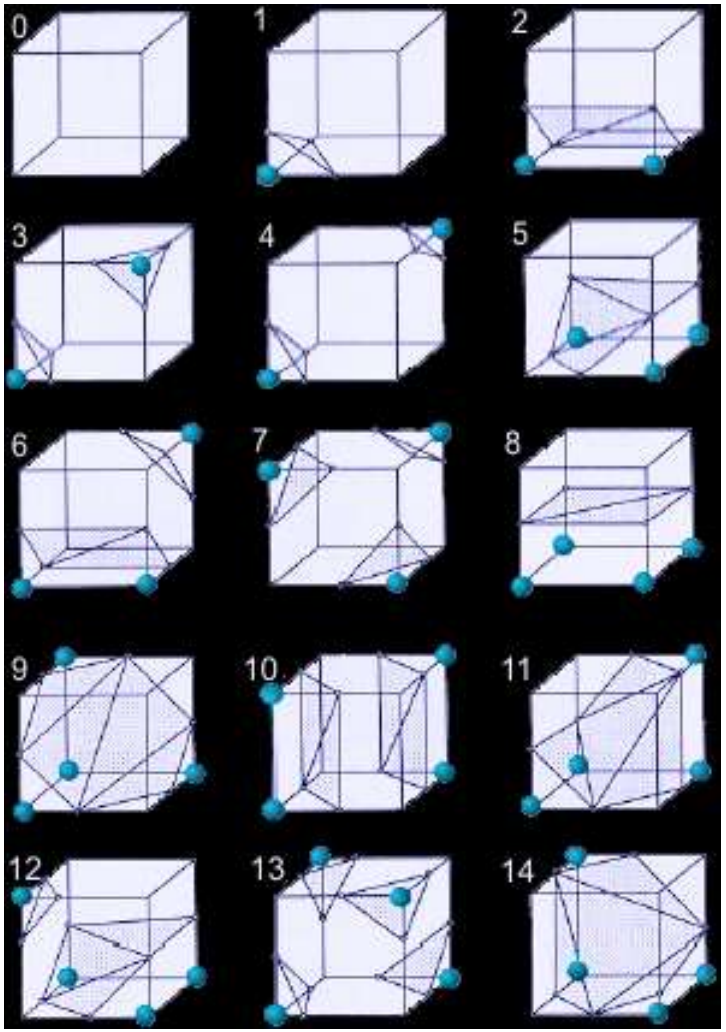
2D - Marching Squares Algorithm:

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:

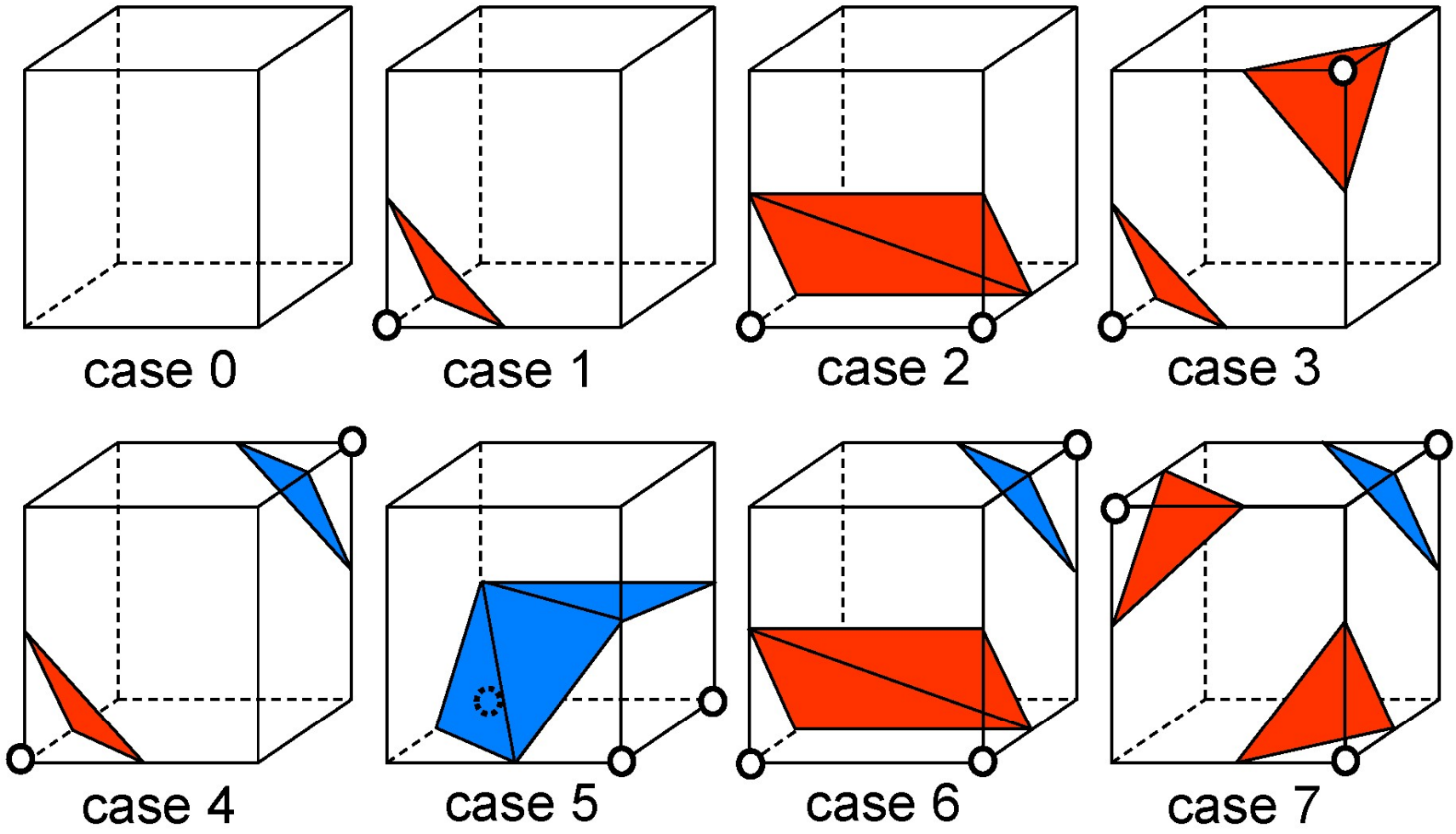
- rotational symmetries of the cube
- reflective symmetries of the cube
- sign changes of $\tilde{f}(x_i)$

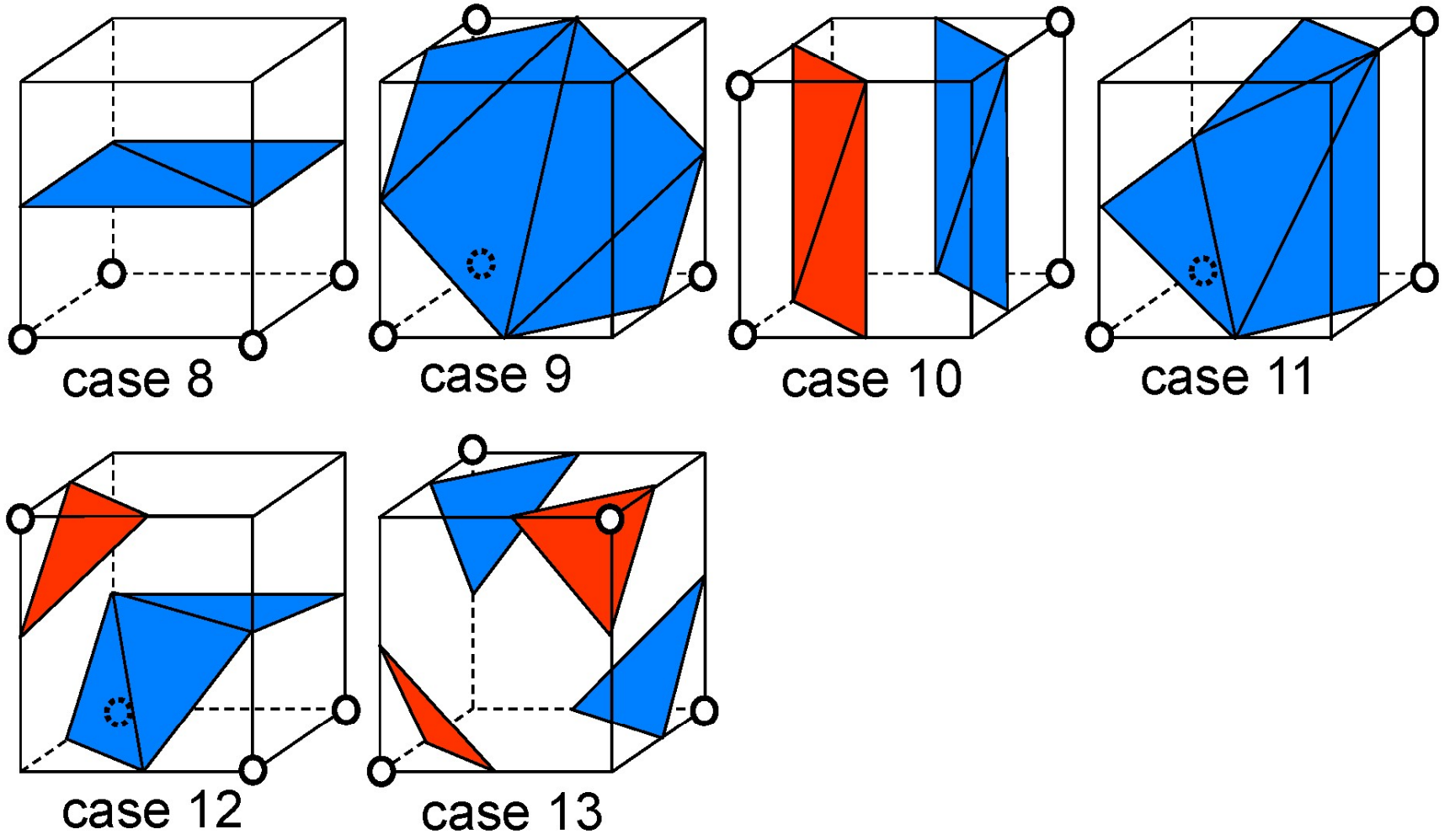
They published a reduced set of 14*) cases shown on the next slides where

- 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

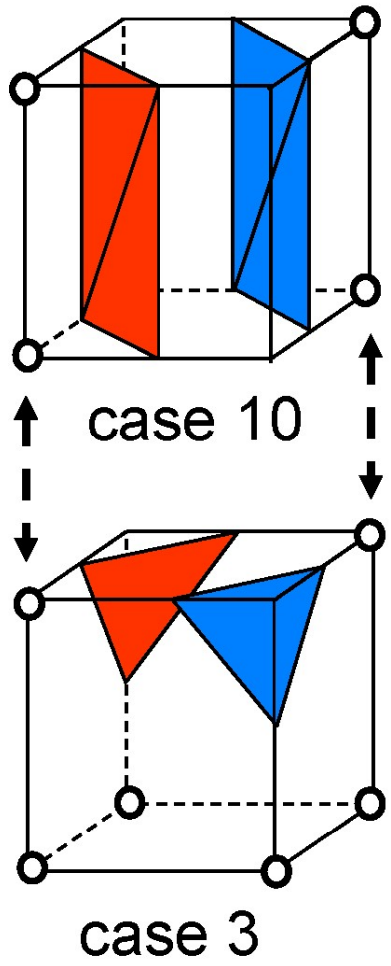
Do the pieces fit together?

- 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.

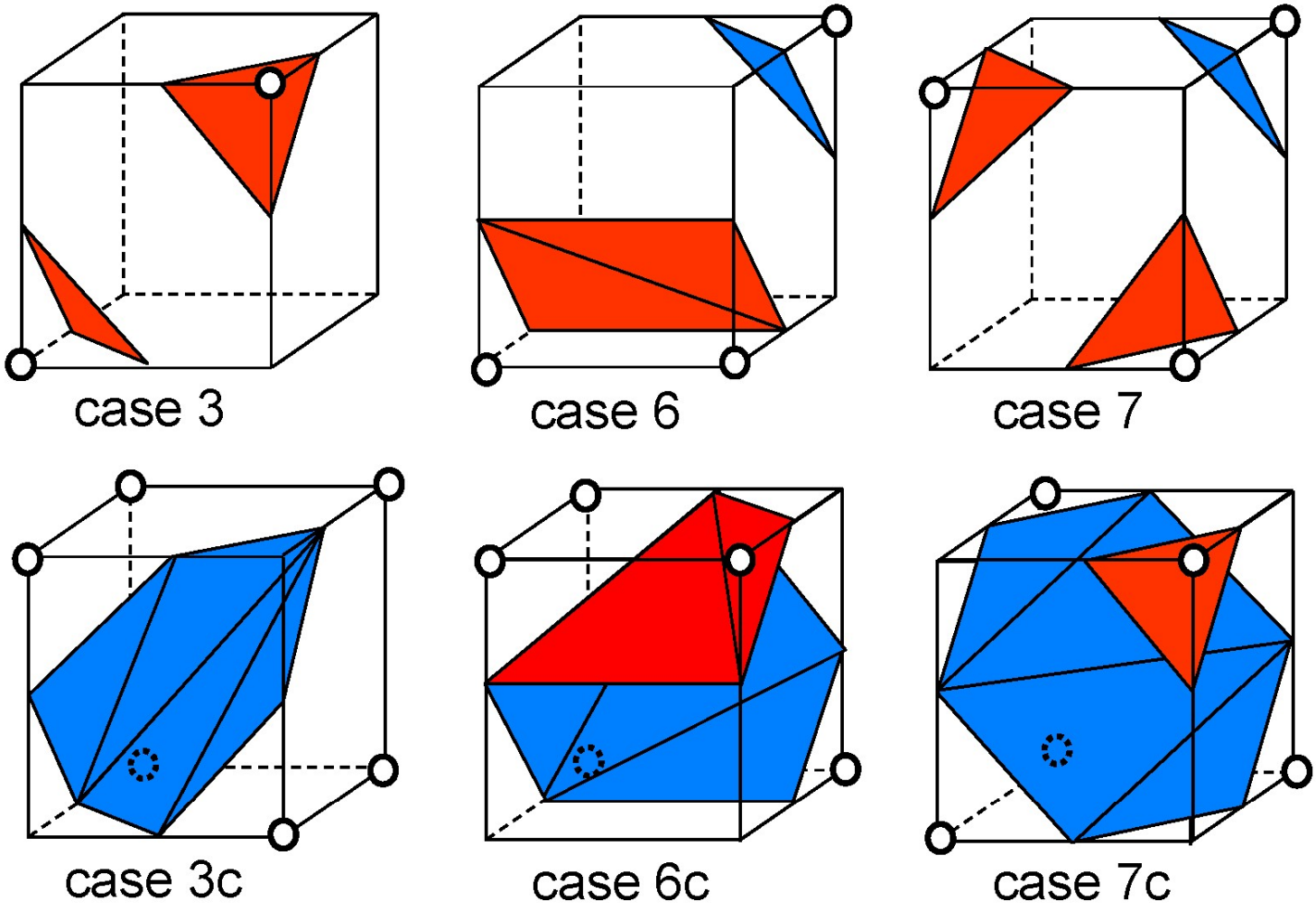
Example

- 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).

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

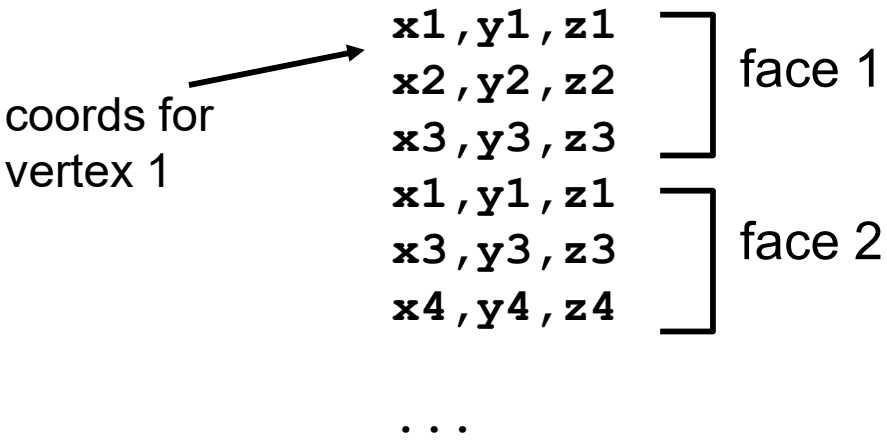
Post-processing steps:

- 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)

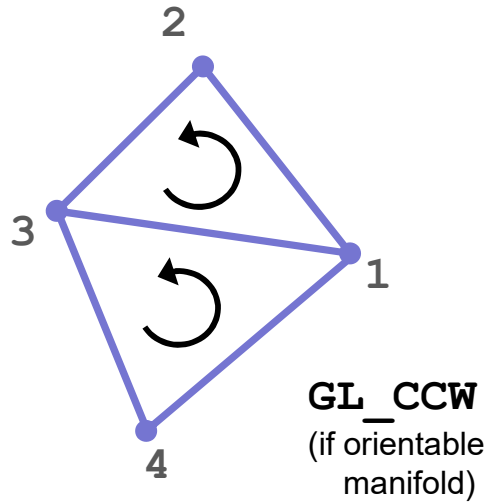
Triangle Mesh Data Structure (1)

Store list of vertices; vertices shared by triangles are replicated

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




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



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

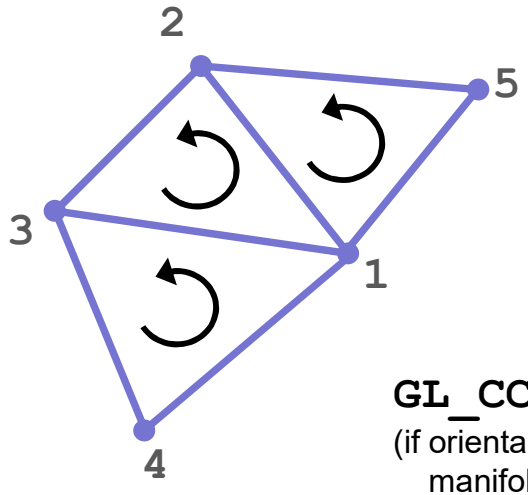
```
x1,y1,(z1)
x2,y2,(z2)
x3,y3,(z3)
x4,y4,(z4)

...
```

face list

```
1,2,3
1,3,4
2,1,5

...
```



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)

Orientability of 2-manifold:

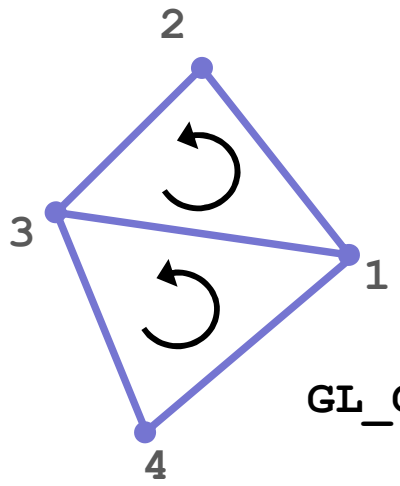
Possible to assign consistent normal vector orientation

not orientable

Moebius strip
(only one side!)

Triangle meshes

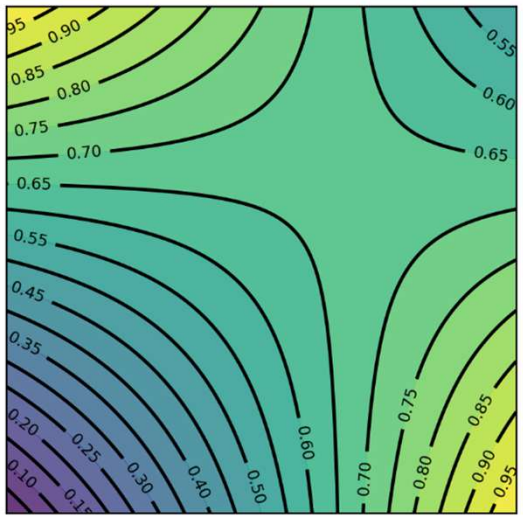
- Edges
  - Consistent ordering of vertices: CCW (counter-clockwise) or CW (clockwise)
    (e.g., (3,1,2) on one side of edge, (1,3,4) on the other side)
- Triangles
  - Consistent front side vs. back side
  - Normal vector; or ordering of vertices (CCW/CW)
  - See also: “right-hand rule”

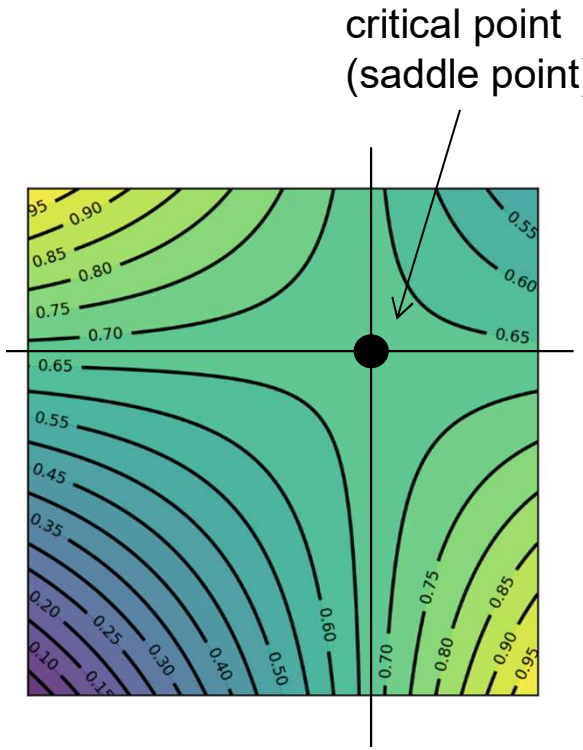


`GL_CCW`

Bi-Linear Interpolation: Critical Points

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





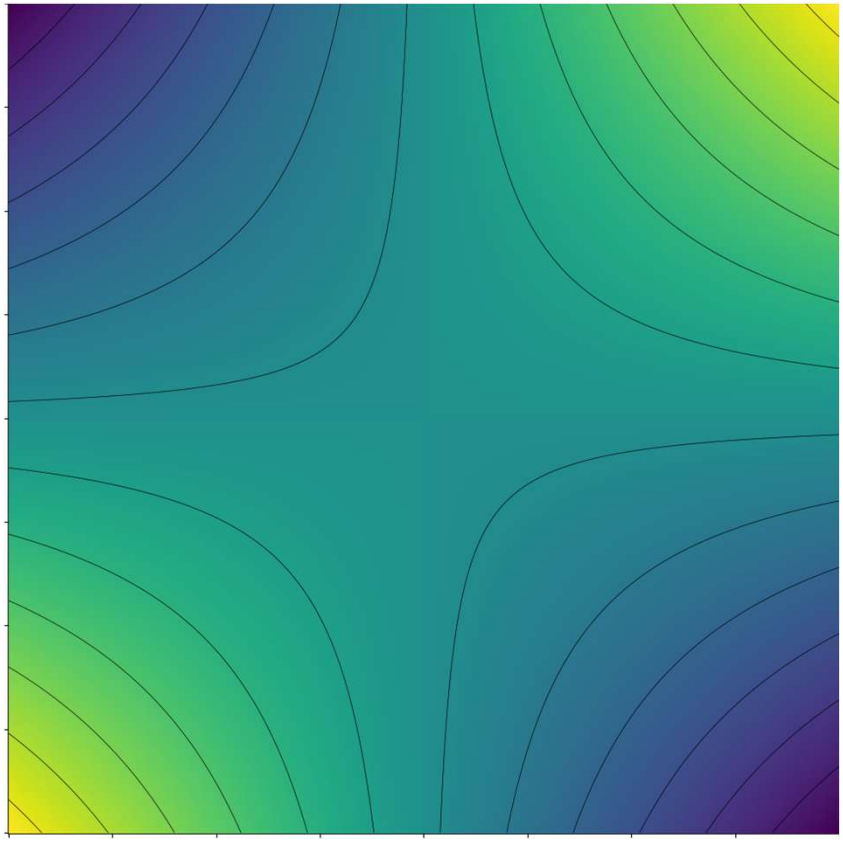
here, the critical value is 2/3=0.666…

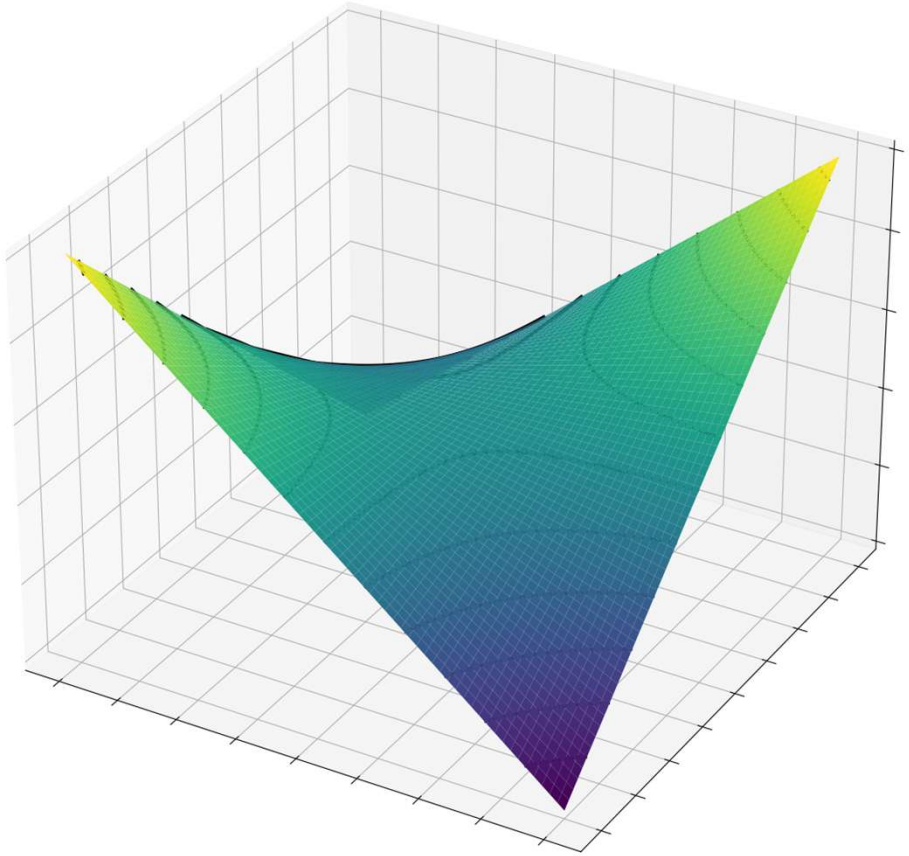
"Asymptotic decider": resolve ambiguous configurations (6 and 9) by comparing specific iso-value with critical value (scalar value at critical point)

Bi-Linear Interpolation

Consider area between 2x2 adjacent samples (e.g., pixel centers)

Example #1: 1 at bottom-left and top-right, 0 at top-left and bottom-right

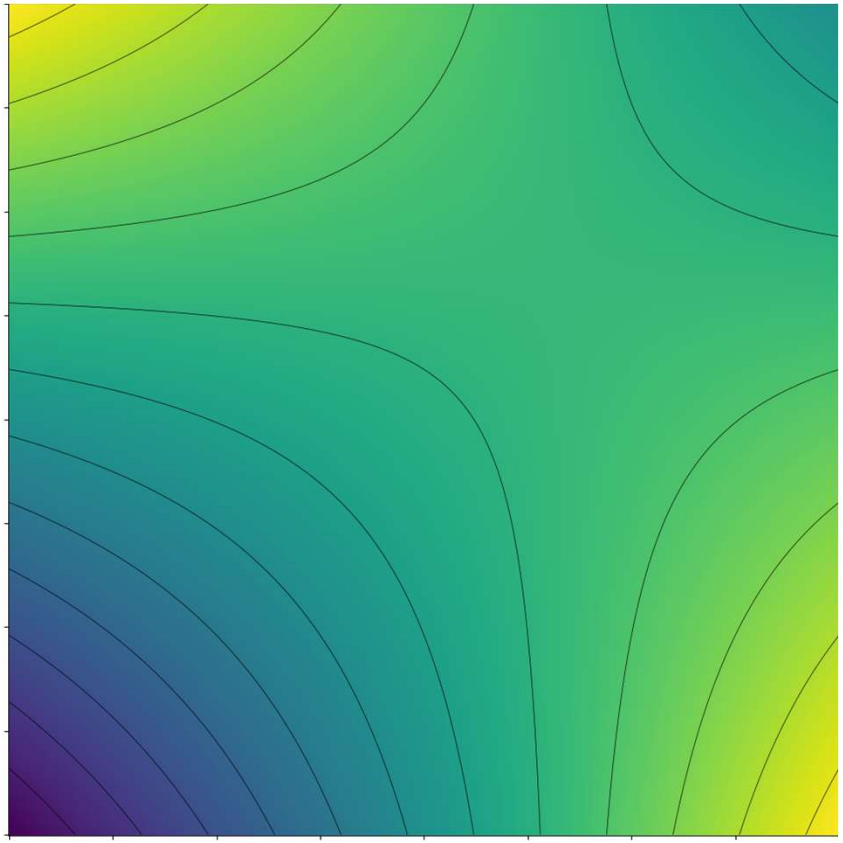


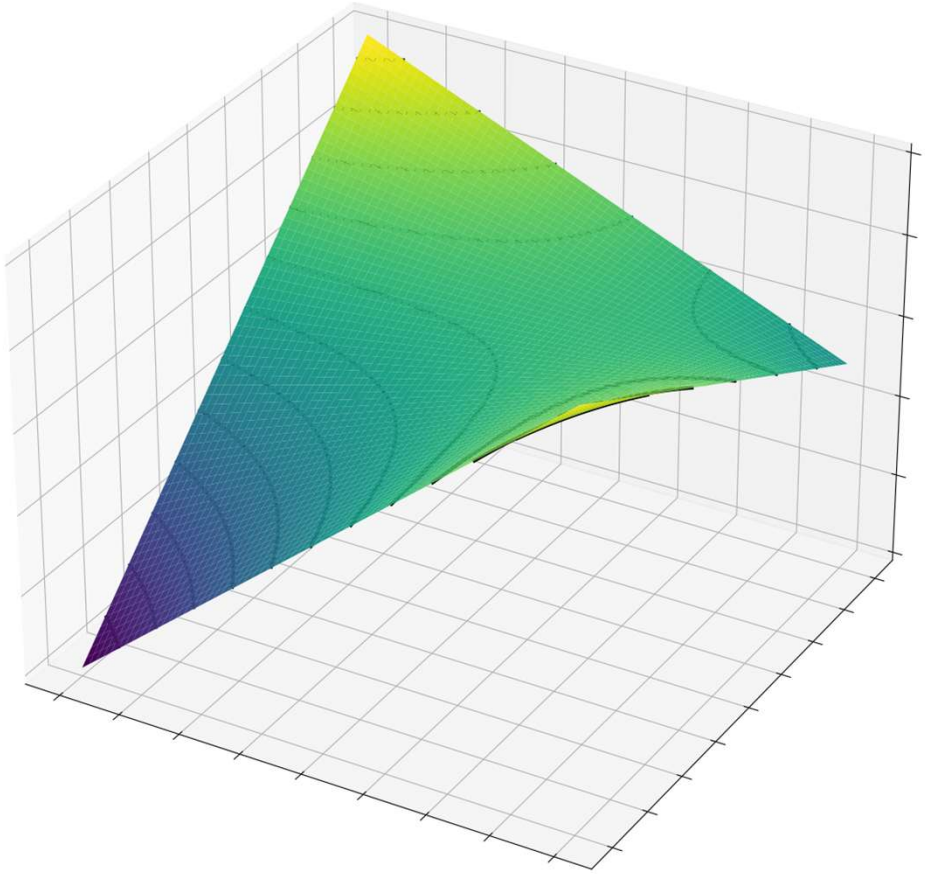


Bi-Linear Interpolation

Consider area between 2x2 adjacent samples (e.g., pixel centers)

Example #2: 1 at top-left and bottom-right, 0 at bottom-left, 0.5 at top-right





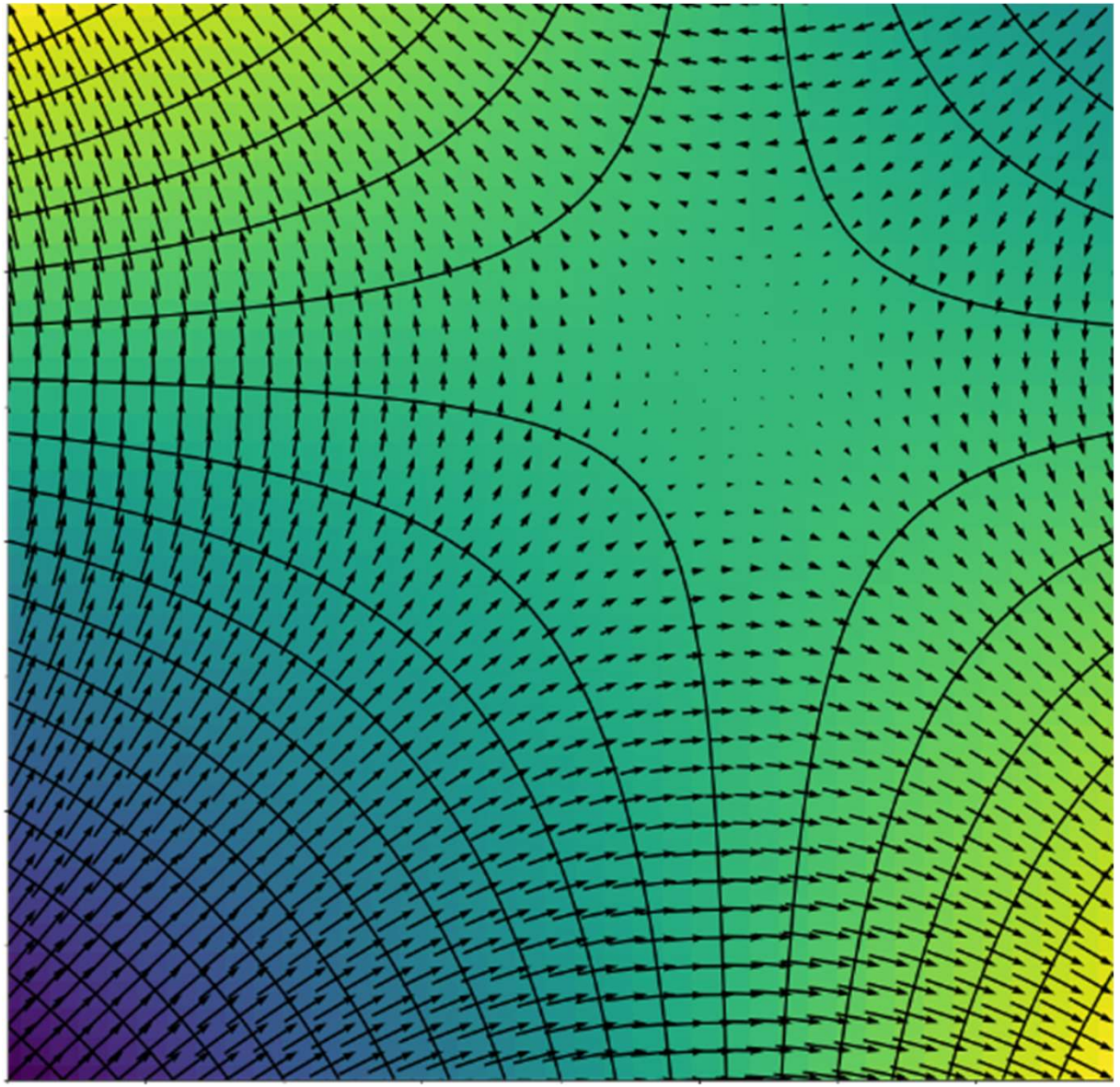
Bi-Linear Interpolation: Critical Points

Compute gradient

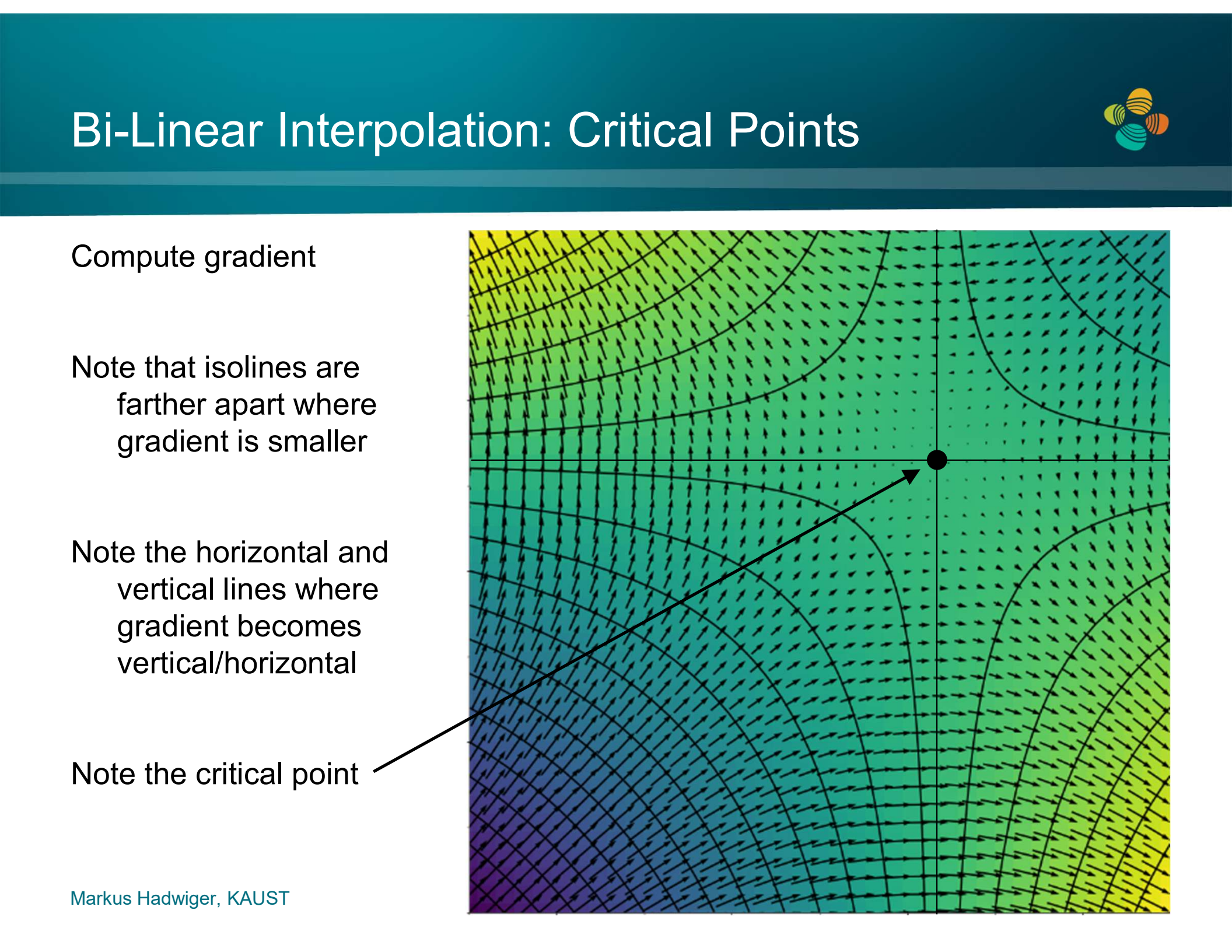
Note that isolines are farther apart where gradient is smaller

Note the horizontal and vertical lines where gradient becomes vertical/horizontal

Note the critical point



Bi-Linear Interpolation: Critical Points



Compute gradient

Note that isolines are farther apart where gradient is smaller

Note the horizontal and vertical lines where gradient becomes vertical/horizontal

Note the critical point

Thank you.

Thanks for material

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