

# **CS 247 – Scientific Visualization**

## **Lecture 10: Scalar Fields, Pt.6**

Markus Hadwiger, KAUST

# Reading Assignment #5 (until Feb 28)



## Read (required):

- Gradients of scalar-valued functions  
<https://en.wikipedia.org/wiki/Gradient>
- Critical points  
[https://en.wikipedia.org/wiki/Critical\\_point\\_\(mathematics\)](https://en.wikipedia.org/wiki/Critical_point_(mathematics))
- Multivariable derivatives and differentials  
[https://en.wikipedia.org/wiki/Total\\_derivative](https://en.wikipedia.org/wiki/Total_derivative)  
[https://en.wikipedia.org/wiki/Differential\\_of\\_a\\_function#Differentials\\_in\\_several\\_variables](https://en.wikipedia.org/wiki/Differential_of_a_function#Differentials_in_several_variables)  
[https://en.wikipedia.org/wiki/Hessian\\_matrix](https://en.wikipedia.org/wiki/Hessian_matrix)
- Dot product, inner product (more general)  
[https://en.wikipedia.org/wiki/Dot\\_product](https://en.wikipedia.org/wiki/Dot_product)  
[https://en.wikipedia.org/wiki/Inner\\_product\\_space](https://en.wikipedia.org/wiki/Inner_product_space)

# From 2D to 3D (Domain)



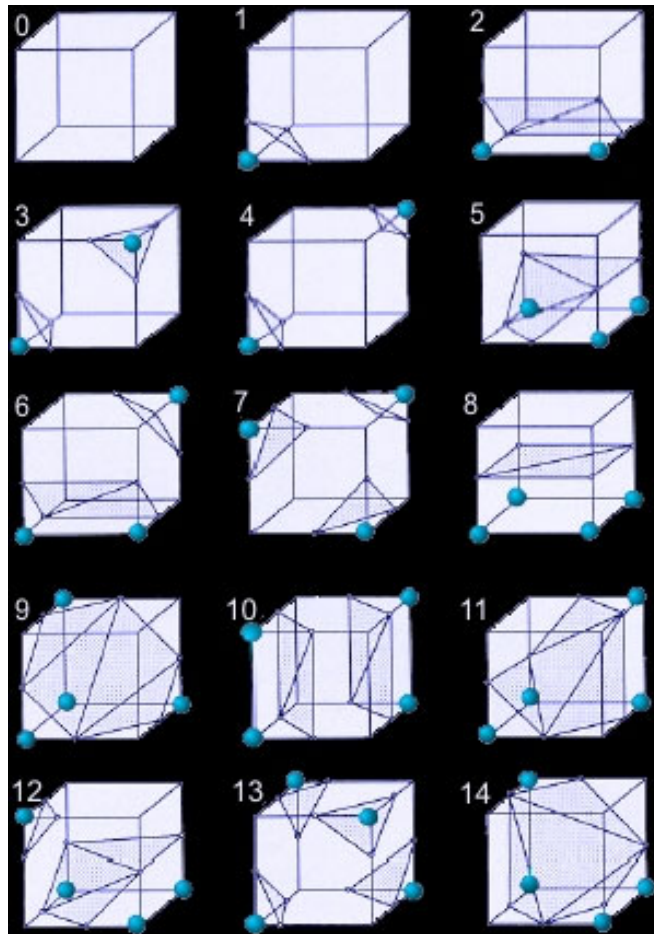
## 2D - Marching Squares Algorithm:

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.

## *The marching cubes algorithm*

How to build up the table of 256 cases?

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

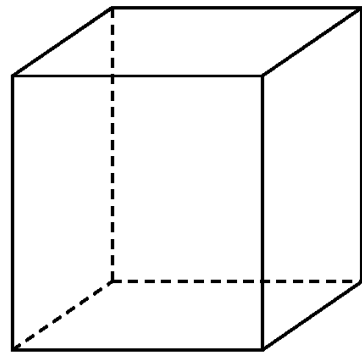
- rotational symmetries of the cube
- reflective symmetries of the cube
- sign changes of  $\tilde{f}(x_i)$

They published a reduced set of 14<sup>\*)</sup> cases shown on the next slides where

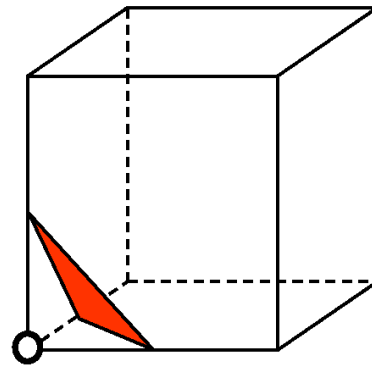
- white circles indicate positive signs of  $\tilde{f}(x_i)$
- the positive side of the isosurface is drawn in red, the negative side in blue.

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

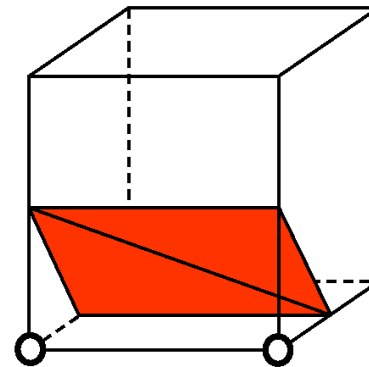
*The marching cubes algorithm*



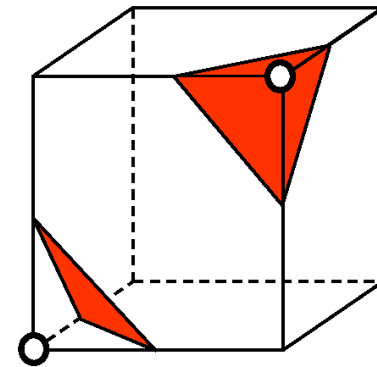
case 0



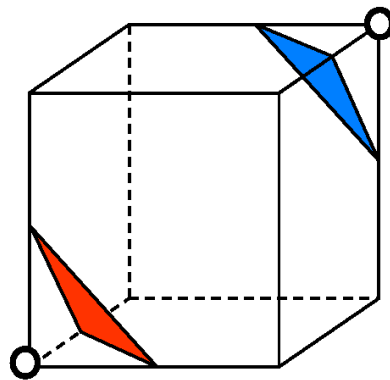
case 1



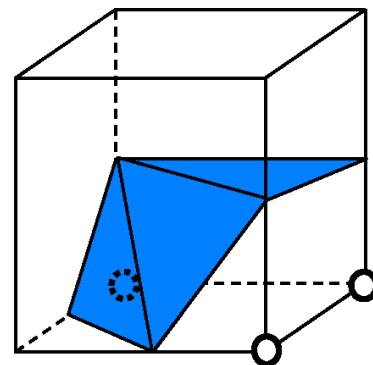
case 2



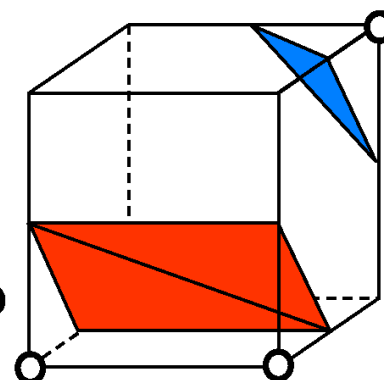
case 3



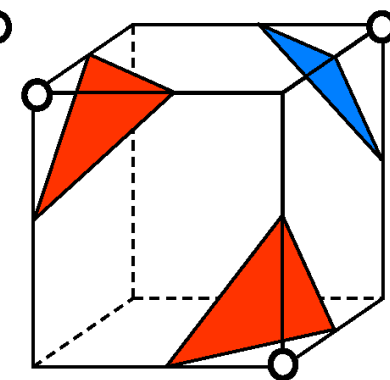
case 4



case 5

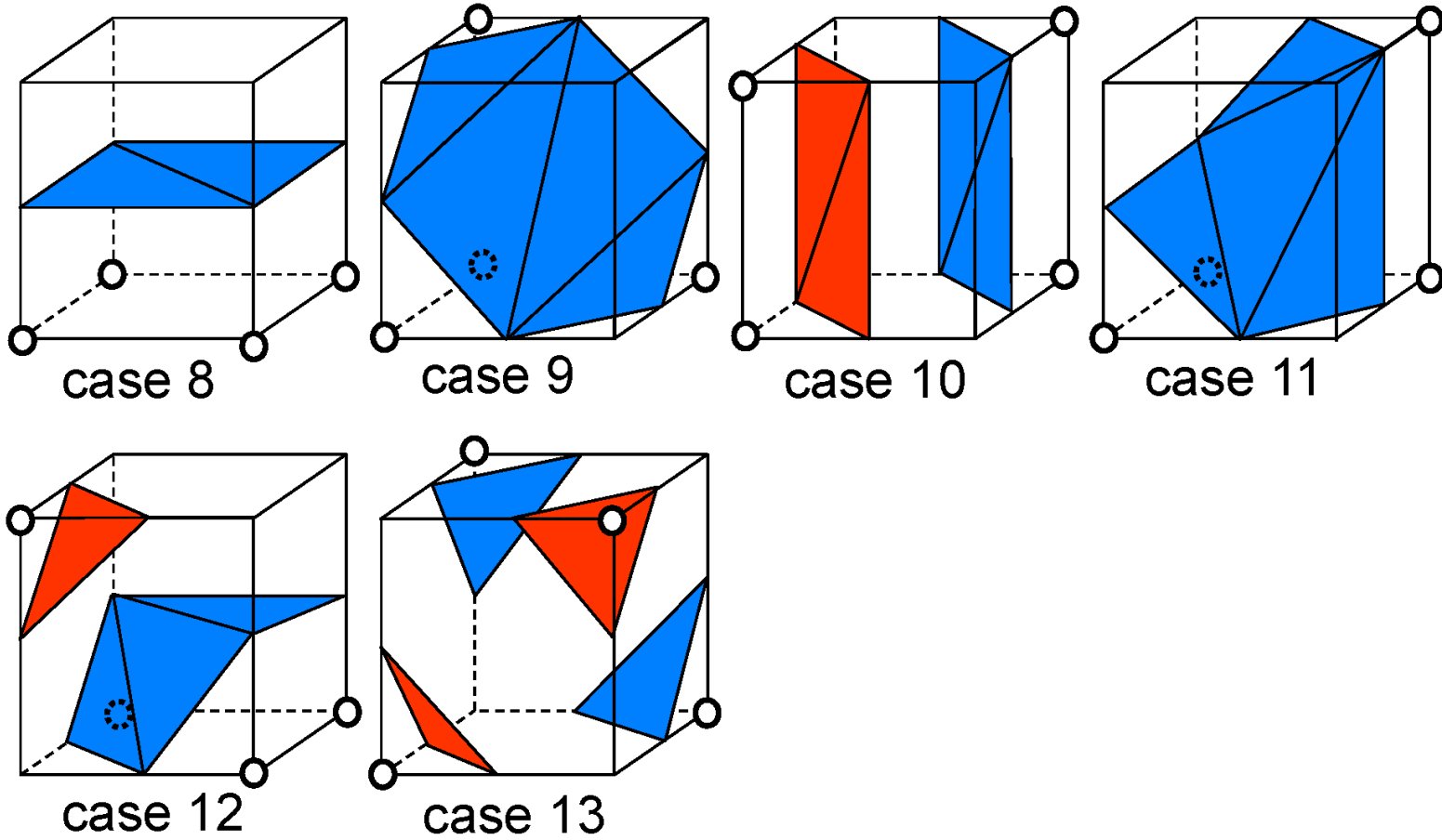


case 6



case 7

*The marching cubes algorithm*





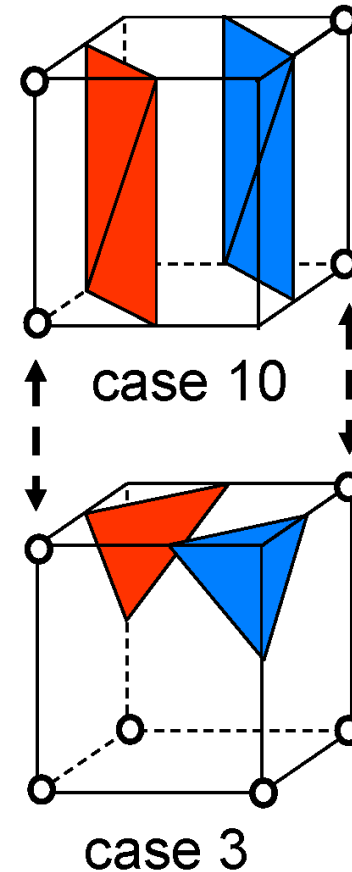
## The marching cubes algorithm

Do the pieces fit together?

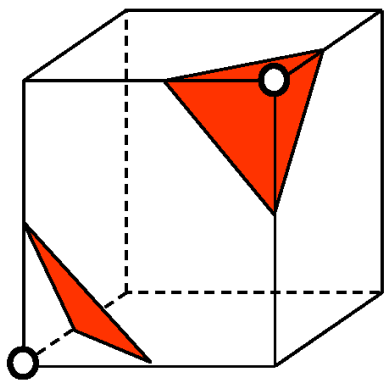
- The correct isosurfaces of the **trilinear interpolant** would fit (trilinear reduces to bilinear on the cell interfaces)
- but the marching cubes polygons don't necessarily fit.

Example

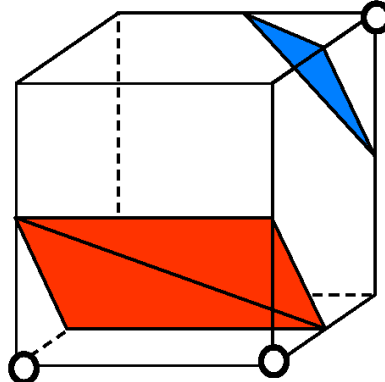
- case 10, on top of
  - case 3 (rotated, signs changed)
- have matching signs at nodes but polygons don't fit.



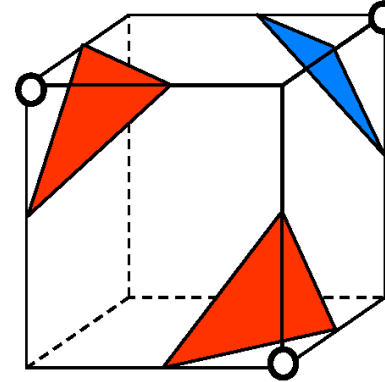
*The marching cubes algorithm*



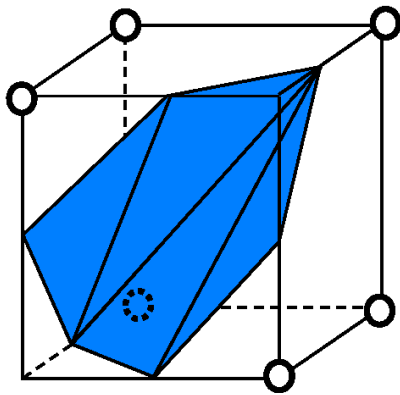
case 3



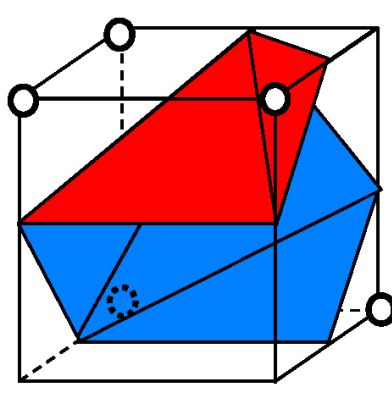
case 6



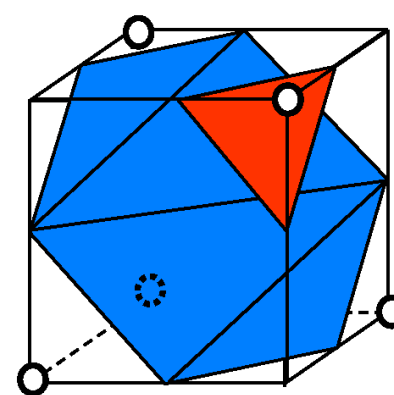
case 7



case 3c



case 6c



case 7c

## *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:

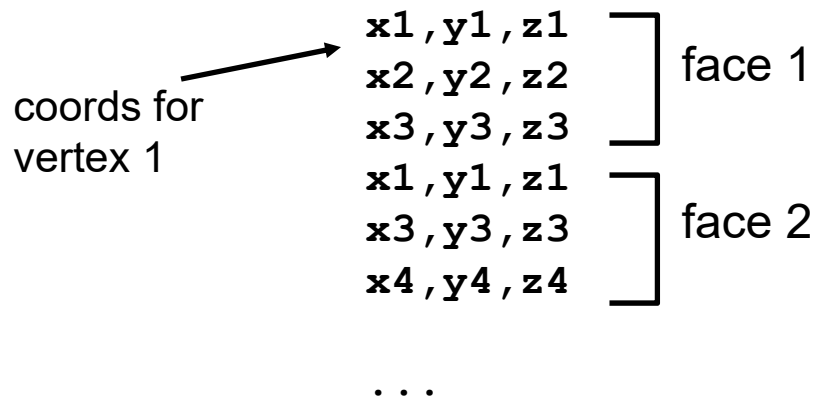
- connect triangles (share vertices)
- compute normal vectors
  - by averaging triangle normals (problem: thin triangles!)
  - by estimating the gradient of the field  $f(x_i)$  (better)

# Triangle Mesh Data Structure (1)

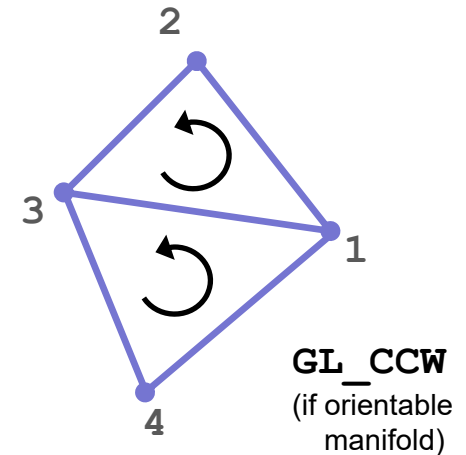


Store list of vertices; vertices shared by triangles are replicated

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



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



Redundant, large storage size, cannot modify shared vertices easily

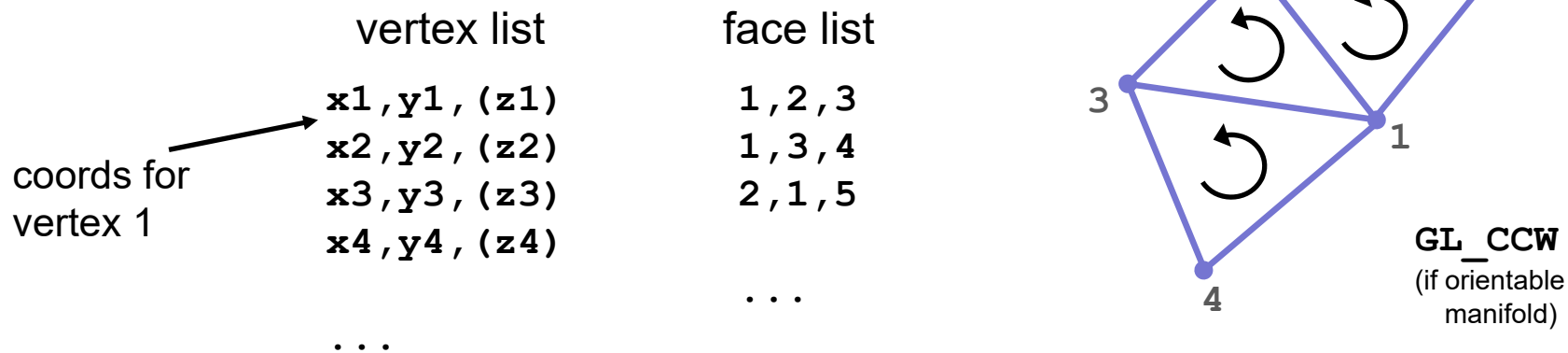
Store data values per face, or separately

# Triangle Mesh Data Structure (2)



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

Render using separate vertex and index arrays / buffers



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

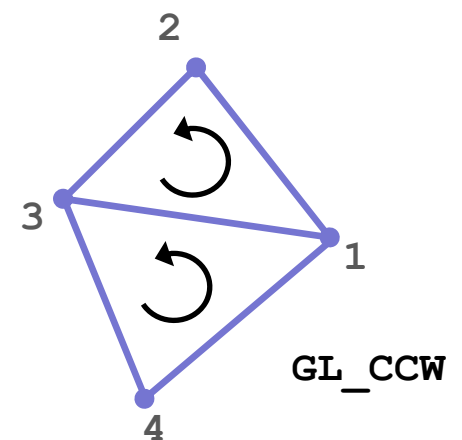
Triangle meshes

- 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”

not orientable



Möbius strip  
(only one side!)



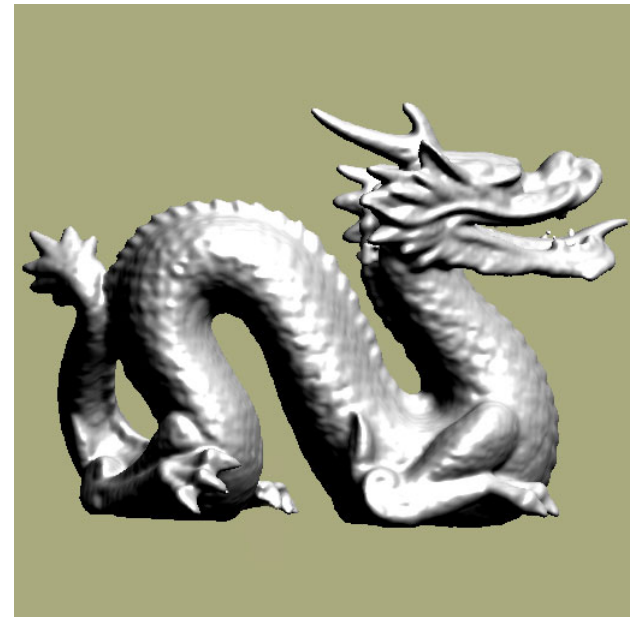
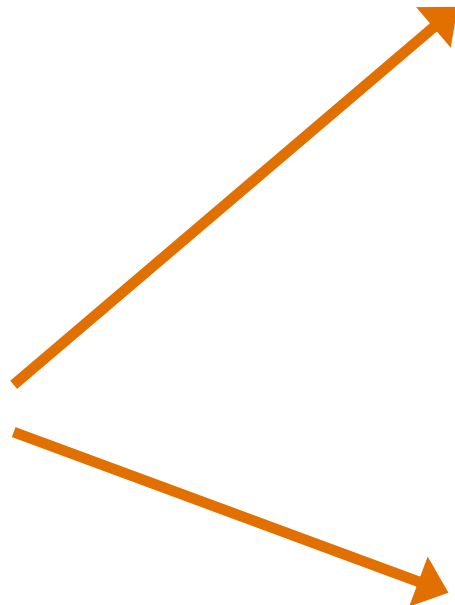


# Iso-Surface / Volume Illumination



# What About Volume Illumination?

Crucial for perceiving shape and depth relationships



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

# Local Illumination in Volumes



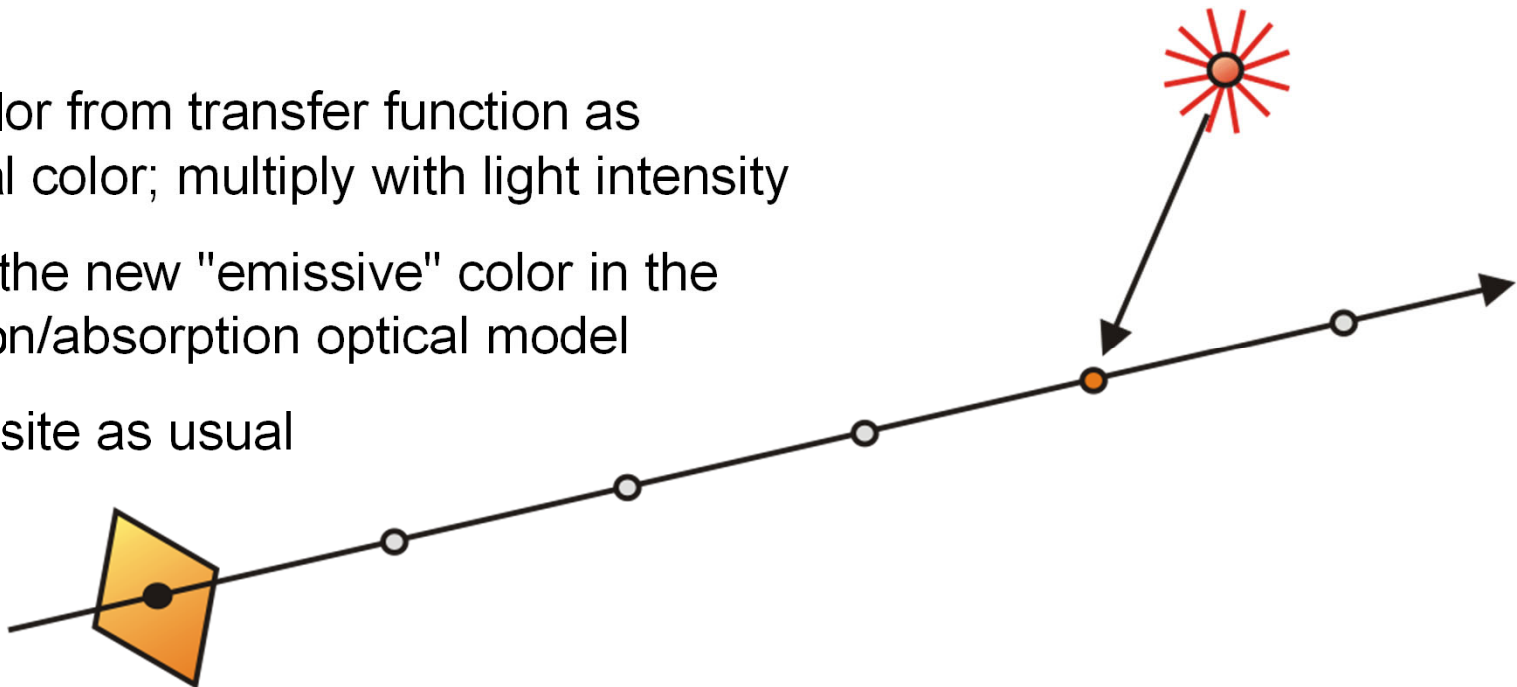
Interaction between light source and point in the volume

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



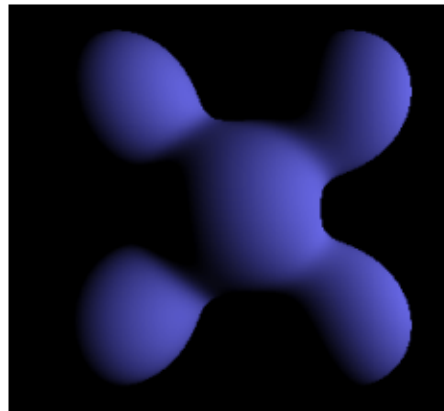
# Local Illumination Model: Phong Lighting Model



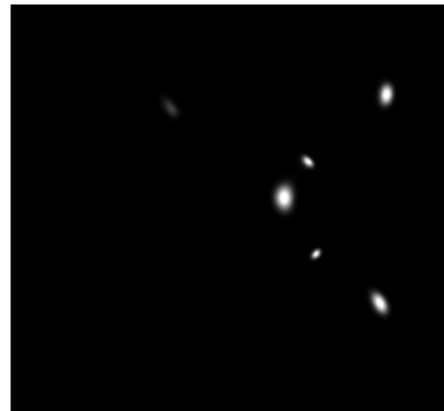
$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$



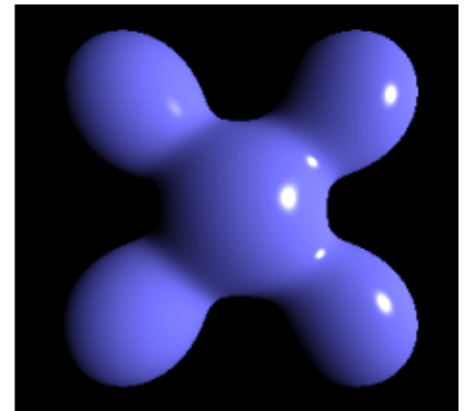
Ambient



Diffuse



Specular

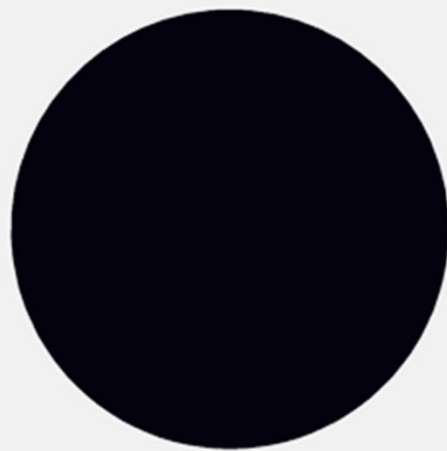


= Phong Reflection

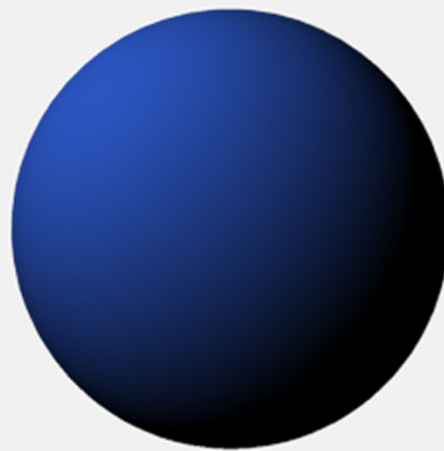
# Local Illumination Model: Phong Lighting Model



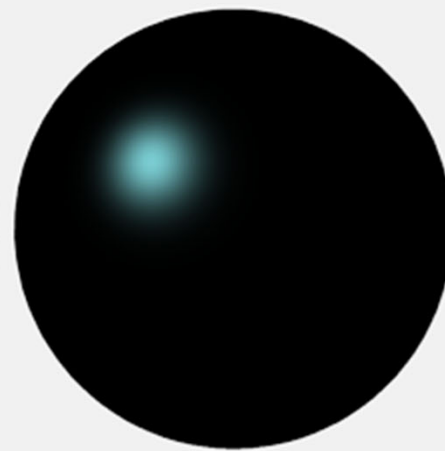
$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$



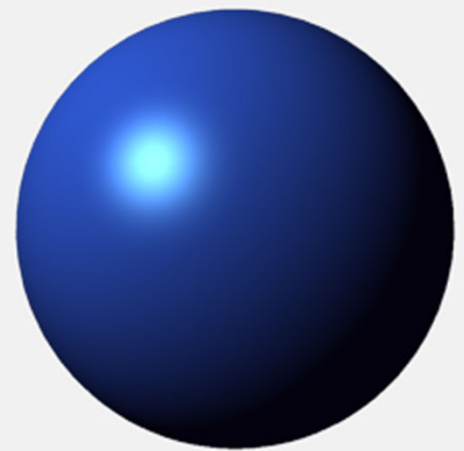
Ambient



Diffuse



Specular



Combined

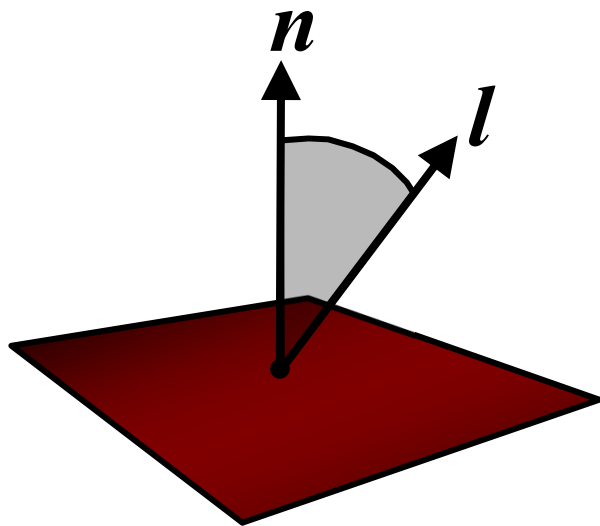
# Local Shading Equations



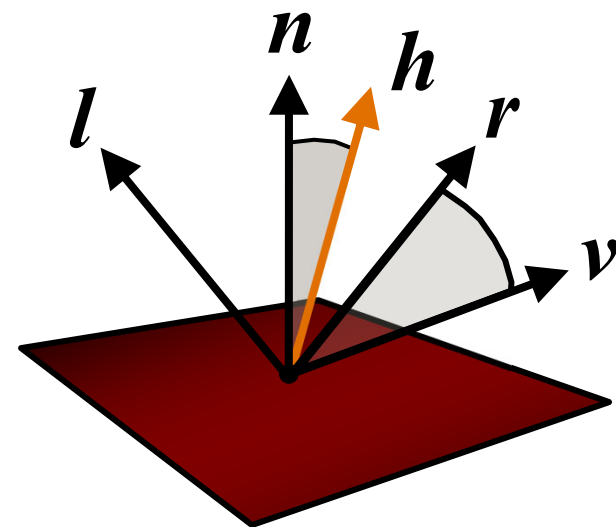
Standard volume shading adapts surface shading

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

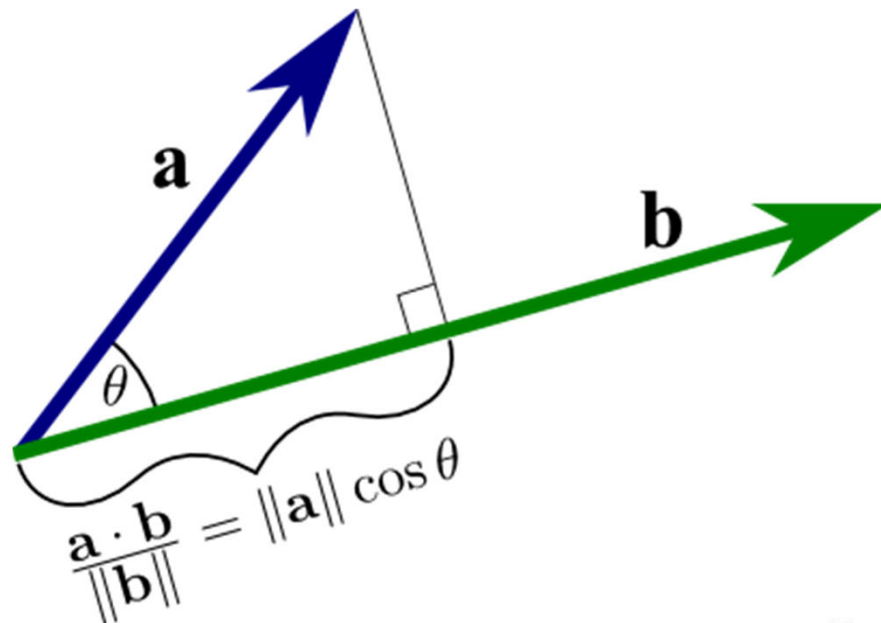
$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i$$

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \theta$$

(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



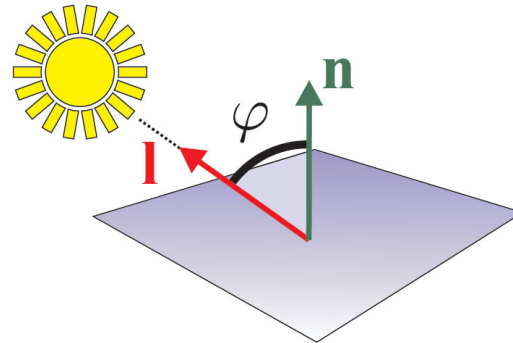
$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$

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

# Local Illumination Model: Phong Lighting Model



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



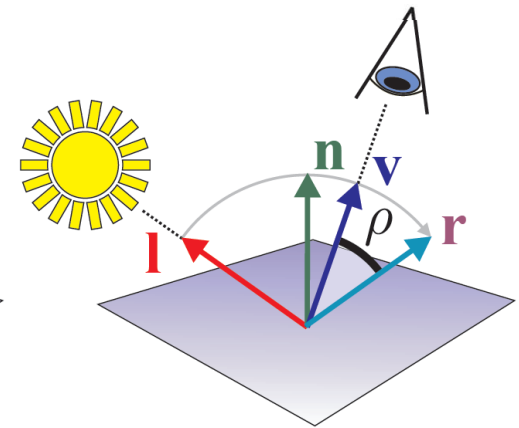
$$\begin{aligned} \mathbf{I}_{\text{diffuse}} &= k_d \mathbf{M}_d \mathbf{I}_d \cos \varphi && \text{if } \varphi \leq \frac{\pi}{2} \\ &= k_d \mathbf{M}_d \mathbf{I}_d \max((\mathbf{n} \cdot \mathbf{l}), 0) \end{aligned}$$



# Local Illumination Model: Phong Lighting Model



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



$$\mathbf{I}_{\text{specular}} = k_s \mathbf{M}_s \mathbf{I}_s \cos^n \rho, \quad \text{if } \rho \leq \frac{\pi}{2}$$

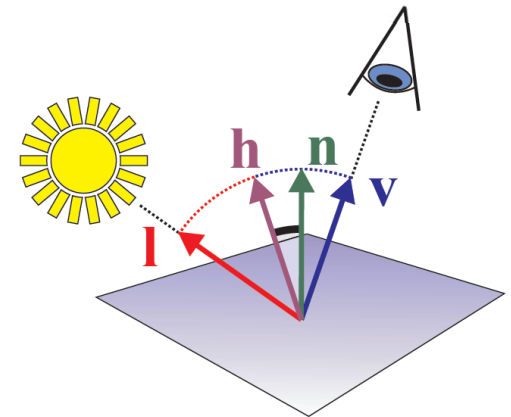
$$= k_s \mathbf{M}_s \mathbf{I}_s (\mathbf{r} \cdot \mathbf{v})^n$$

must also clamp!

# Local Illumination Model: Phong Lighting Model



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



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

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

must also clamp!  
half-way vector

# The Gradient as Normal Vector



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

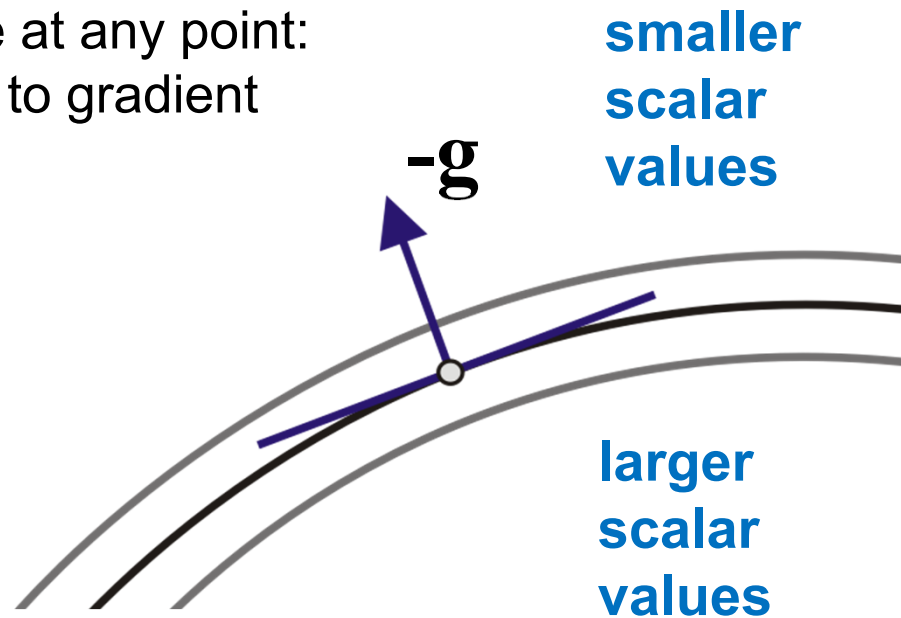
$$\mathbf{g} = \nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)^T$$

(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)

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



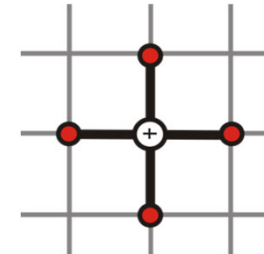
# (Numerical) Gradient Reconstruction



We need to reconstruct the derivatives of a continuous function given as discrete samples

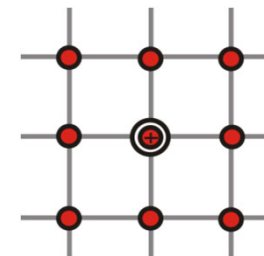
## Central differences

- Cheap and quality often sufficient ( $2 \times 3$  neighbors in 3D)



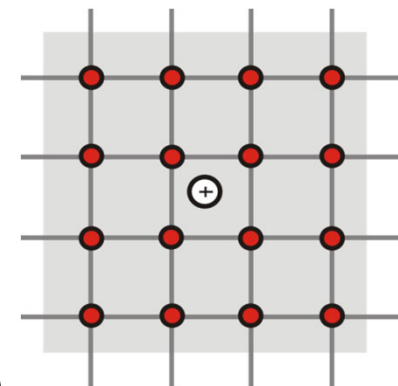
## Discrete convolution filters on grid

- Image processing filters; e.g. Sobel ( $3^3$  neighbors in 3D)



## Continuous convolution filters

- Derived continuous reconstruction filters
- E.g., the cubic B-spline and its derivatives ( $4^3$  neighbors)



# Finite Differences



Obtain first derivative from Taylor expansion

$$\begin{aligned} f(x_0 + h) &= f(x_0) + \frac{f'(x_0)}{1!} h + \frac{f''(x_0)}{2!} h^2 + \dots \\ &= \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} h^n. \end{aligned}$$

Forward differences / backward differences

$$f(x_0)' = \frac{f(x_0 + h) - f(x_0)}{h} + o(h)$$

$$f(x_0)' = \frac{f(x_0) - f(x_0 - h)}{h} + o(h)$$

# Finite Differences



## Central differences

$$f(x_0 + h) = f(x_0) + \frac{f'(x_0)}{1!} h + \frac{f''(x_0)}{2!} h^2 + o(h^3)$$

$$f(x_0 - h) = f(x_0) - \frac{f'(x_0)}{1!} h + \frac{f''(x_0)}{2!} h^2 + o(h^3)$$

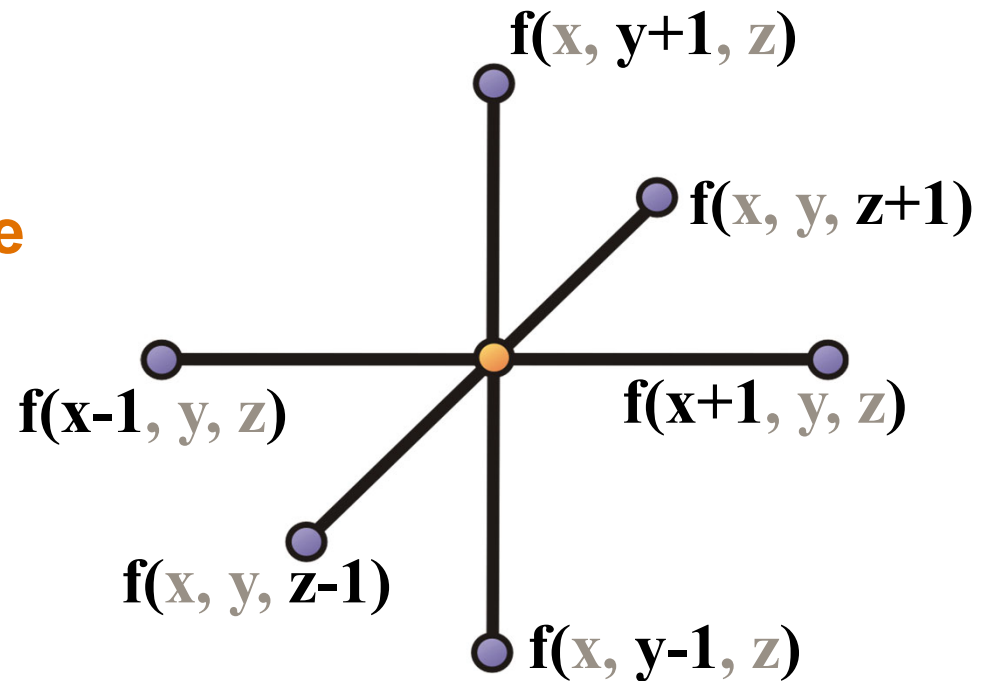
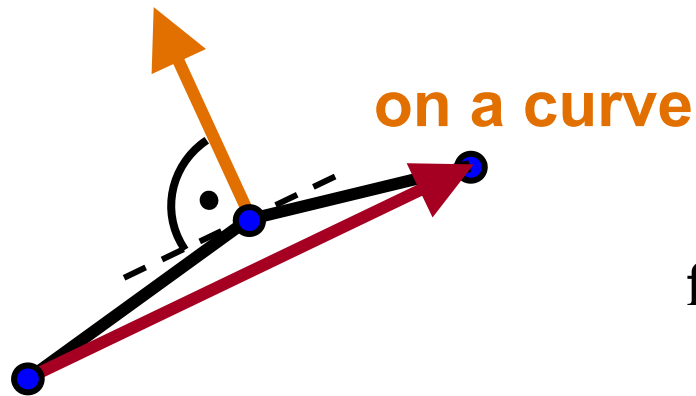
$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + o(h^2)$$

# Central Differences



Need only two neighboring voxels per derivative

Most common method



$$g_x = 0.5 ( f(x+1, y, z) - f(x-1, y, z) )$$

$$g_y = 0.5 ( f(x, y+1, z) - f(x, y-1, z) )$$

$$g_z = 0.5 ( f(x, y, z+1) - f(x, y, z-1) )$$

in a volume

# Thank you.

## Thanks for material

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