

Computer Graphics: Specialized Modeling Techniques

**Dr. Michael Laszlo
Nova Southeastern University
School of Computer and Information Sciences
2001**

Specialized Modeling and Visualization Techniques

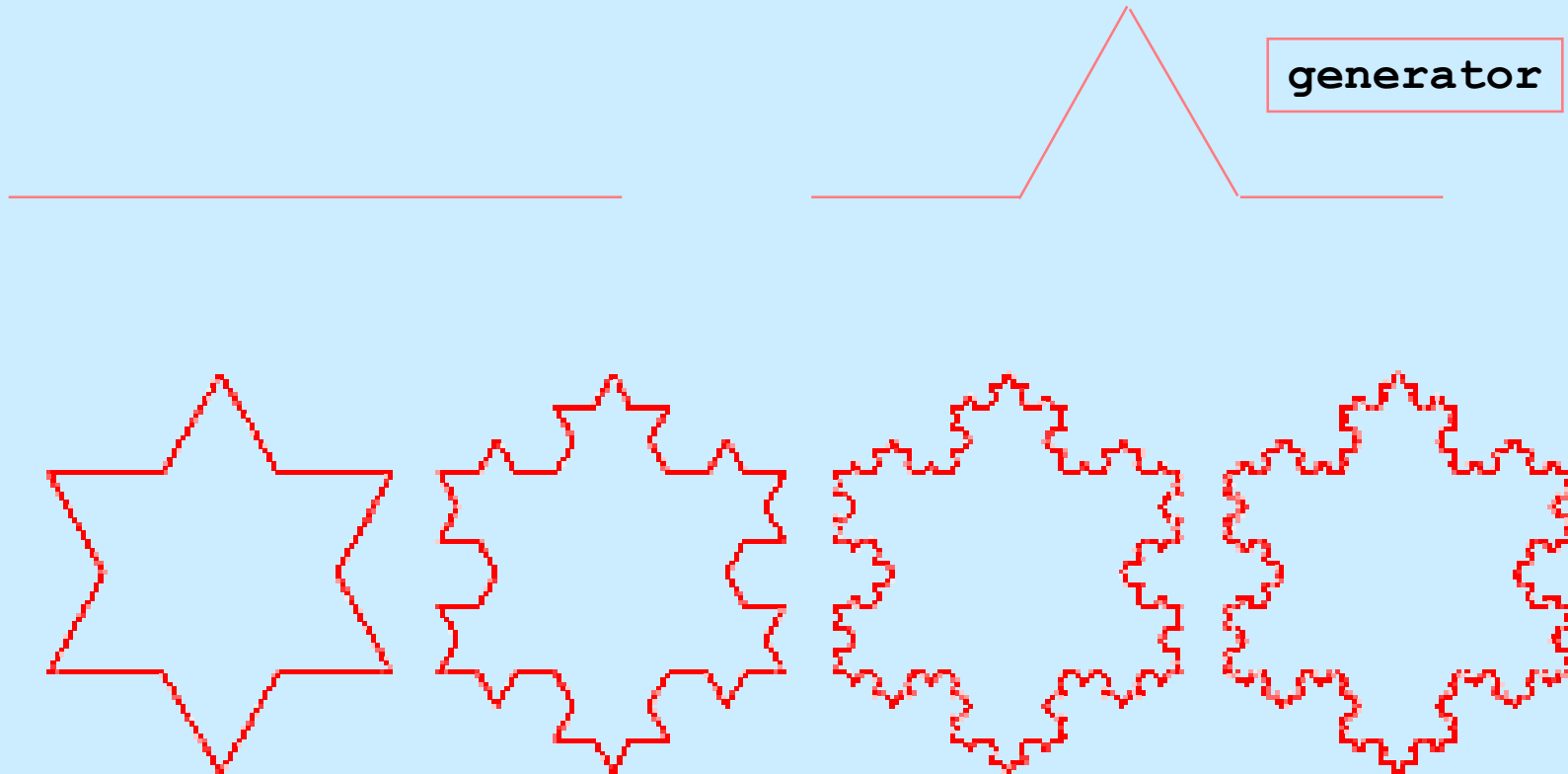
- This slide set covers three specialized modeling and visualization techniques:
 - Fractal modeling
 - Volume rendering: marching cubes
 - Volume rendering: volume ray tracing

Fractals

- *Fractals* were popularized by the French mathematician Benoit Mandelbrot in the 1970s, and have since been adapted to many purposes.
- A *fractal* is an object possessing a substantial measure of exact or statistical *self-similarity* at various resolutions, although the strict definition requires self-similarity at all resolutions. Self-similarity means simply that the object is similar to itself at different resolutions. Thus we can view the rough, jagged shape of a mountain; and when we zoom in on a crag, we find it to possess a similar rough jaggedness; and when we zoom in to a rock that is part of the crag, we again find this same texture. Similarly, when we focus on a coastline at various resolutions, we find bays and peninsulas that possess a tendency to squiggle at random. Many natural phenomena are fractal, and indeed, fractal modeling is often used in computer graphics to model natural phenomena.

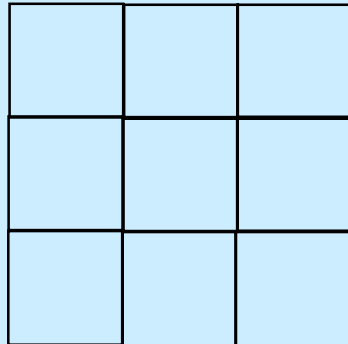
Fractals

- Exact self-similarity can be observed in the *Koch snowflake*, in which an initial pattern—a line—is recursively replaced by a pattern called a *generator*.



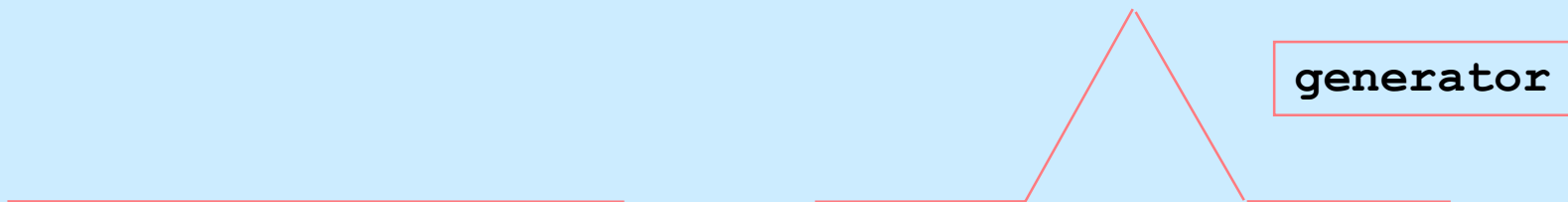
Fractals

- Associated with self-similarity is the idea of *fractal dimension*. Euclidean objects possess integral dimension: a line is 1-dimensional, a disk is 2-dimensional, and a solid ball 3-dimensional. Fractals, on the other hand, possess *fractional* dimension.
- If we take a line and divide it into N equal parts, each part looks like the original line scaled down by a factor of $N = N^{1/1} = N^{1/d}$ (where dimension $d = 1$).
- One dimension higher, if we take a filled square and divide it into N equal parts, each part looks like the original scaled down by a factor of $N^{1/2} = N^{1/d}$ (where $d = 2$).



Fractals

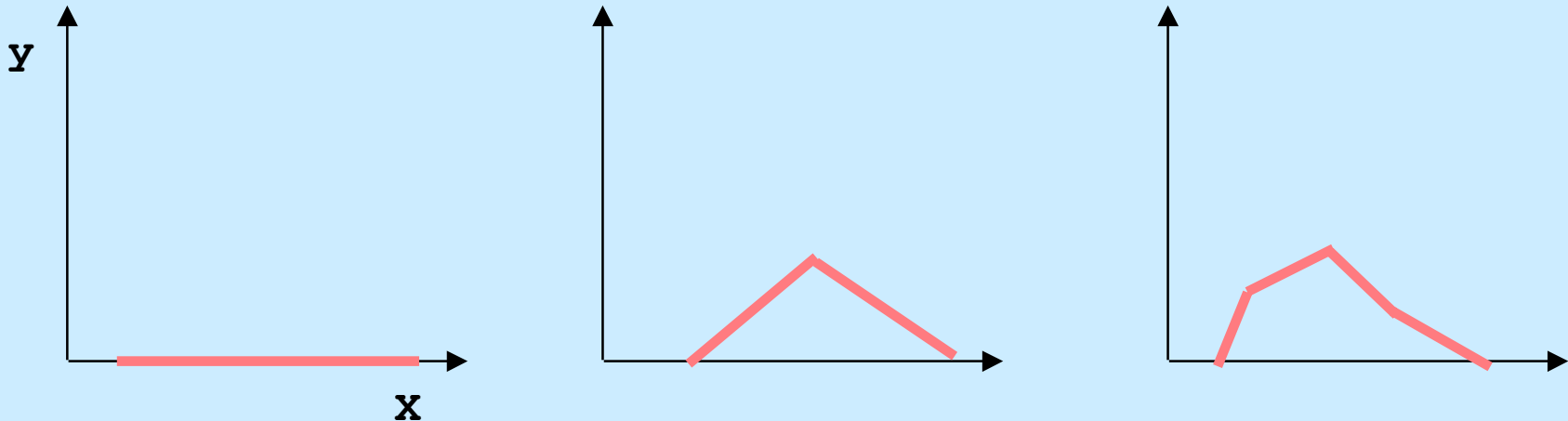
- Suppose we take a Koch snowflake and divide it into 4 equal parts. Now each part looks like the original scaled down by a factor of 3, hence we have $4^{1/d} = 3$:
$$4^{1/d} = 3$$
$$4 = 3^d$$
$$d \log 3 = \log 4$$
$$d = \log 4 / \log 3 = 1.2619$$
- Hence the Koch snowflake has fractal dimension about 1.2619. The length of a Koch snowflake depends on the length of the ruler (unit of measure). The shorter the ruler, the greater the length of the Koch snowflake. The same holds true of coastlines, since coastlines are also fractal.



Fractals

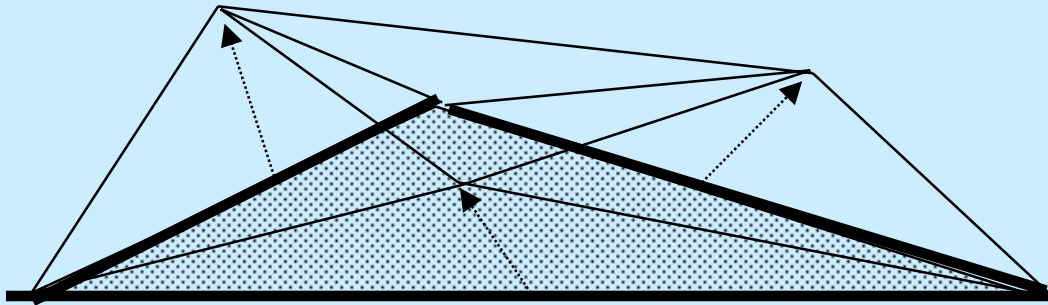
- A class of *fractal mountains* can be generated by recursive subdivision. Such mountains possess statistical self-similarity. In 1 dimension, we start with a line segment and recursively find the current line segment's midpoint and perturb its position, thereby producing two line segments. The same perturbation function P is always used, a function of the line segment's length, scaled by a random number R drawn from the same distribution:

$$\begin{aligned}x' &= 0.5 (x_i + x_{i+1}) \\y' &= 0.5 (y_i + y_{i+1}) + P(x_{i+1} - x_i) R(x')\end{aligned}$$



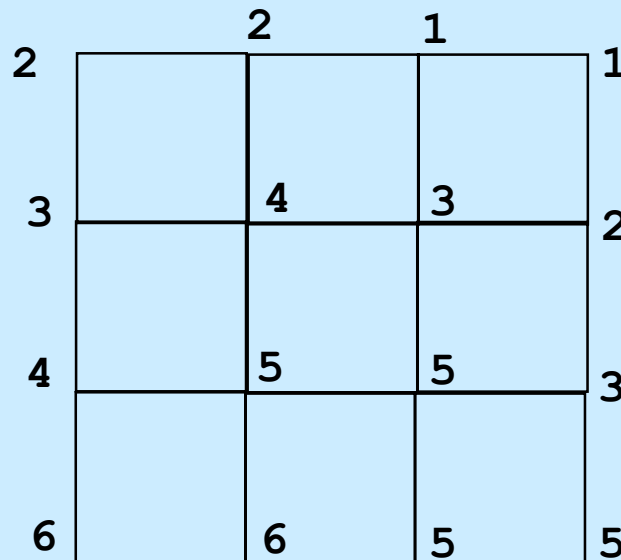
Fractals

- To construct a fractal mountain one dimension higher, we work with triangulations. The midpoints of the 3 edges of a triangle are perturbed, then connected by edges. Thus the original triangle is decomposed into 7 non-coplanar triangles.



Density Fields

- A *density field* is a field of scalar values (real numbers) defined within a block of space. One way to represent a density field is to subdivide the space into a uniform grid of cubes and associate a value with every lattice point (the corner points of the cubes). Interpolation is then used to obtain density values at points interior to cubes.
- 3D density fields are often obtained by volumetric simulation and imaging techniques. For example, medical tomographic imaging yields a 3D density field corresponding to tissue density. Density fields obtained in this way are sometimes called *3D digital images* and *3D data grids*.



A density field
in two dimensions.

Density Fields

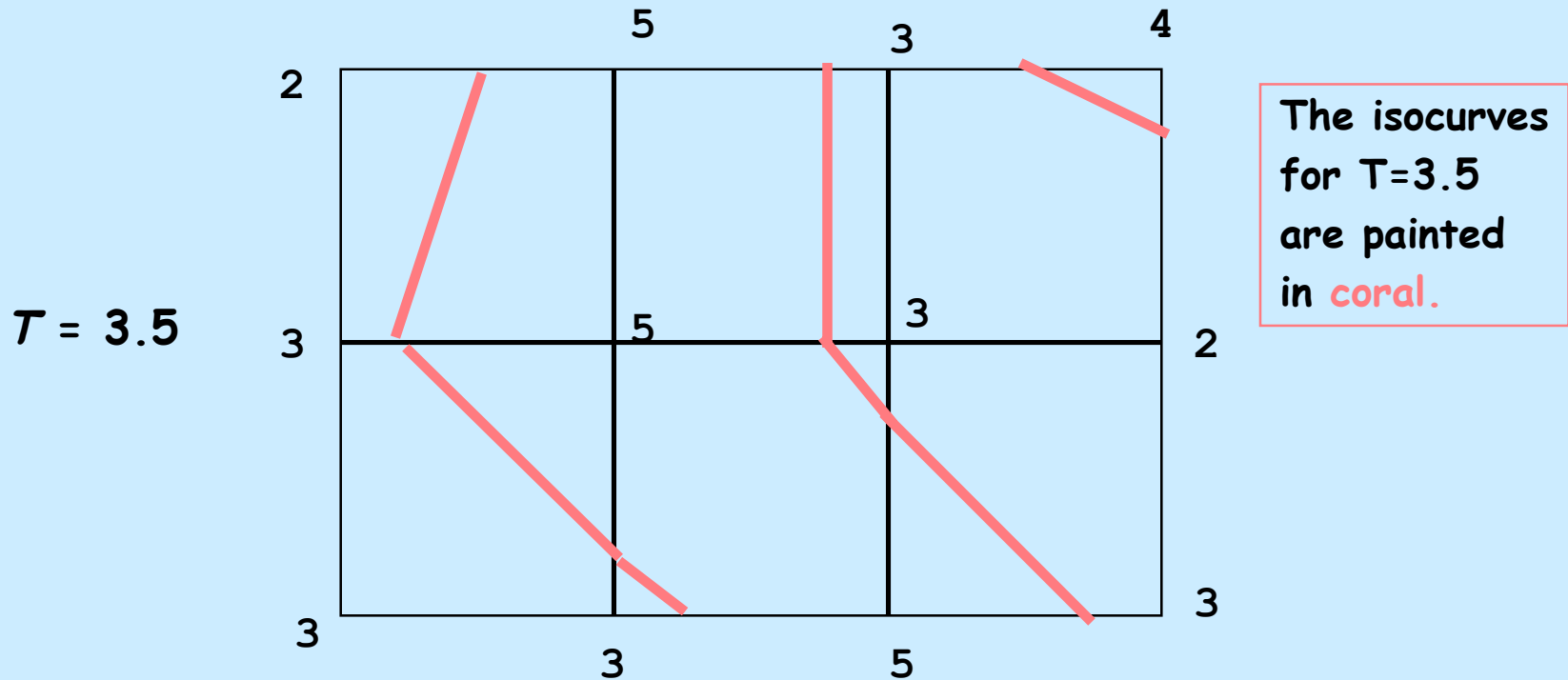
- A surface exists wherever the density is of some constant value, called a *threshold*. The surface, called an *isosurface*, is analogous to the isobars of a weather map, one dimension higher. An isosurface may be empty, connected, or disconnected (i.e., a collection of disjoint connected surfaces).
- In two dimensions, we have curves of constant value, which are called isocurves or isobars. For example, isobars are depicted on weather maps showing temperatures; the isobars are curves of constant temperature 70° , 60° , 50° , and so on.
- In three dimensions, we might obtain a density field from medical imaging of, say, a person's head. The threshold can be set to a value that lies between the density of bone and the density of brain tissue. The resulting isosurface corresponds to the brain-skull surface.

Volume Rendering

- *Volume rendering* is used to explore the interior of a 3D (or higher dimensional) solid using 2D images. There exist many applications for volume rendering: interpreting the features in a 3D image obtained via medical scanning, exploring 3D (or higher dimensional) data sets obtained from simulation studies, viewing pressure/temperature/humidity readings obtained from a block of the atmosphere, data from medical imaging, and so on.
- A 3D digital image stores a value in each *voxel*. We may treat such an image as a data grid in which a real number is associated with the corners of the subcubes, as described in the previous slides.
- We use metaball modeling as an artistic technique, sculpting the data grid in order to evoke certain forms. In contrast, we use volume rendering to *explore* a given data grid without modifying it.

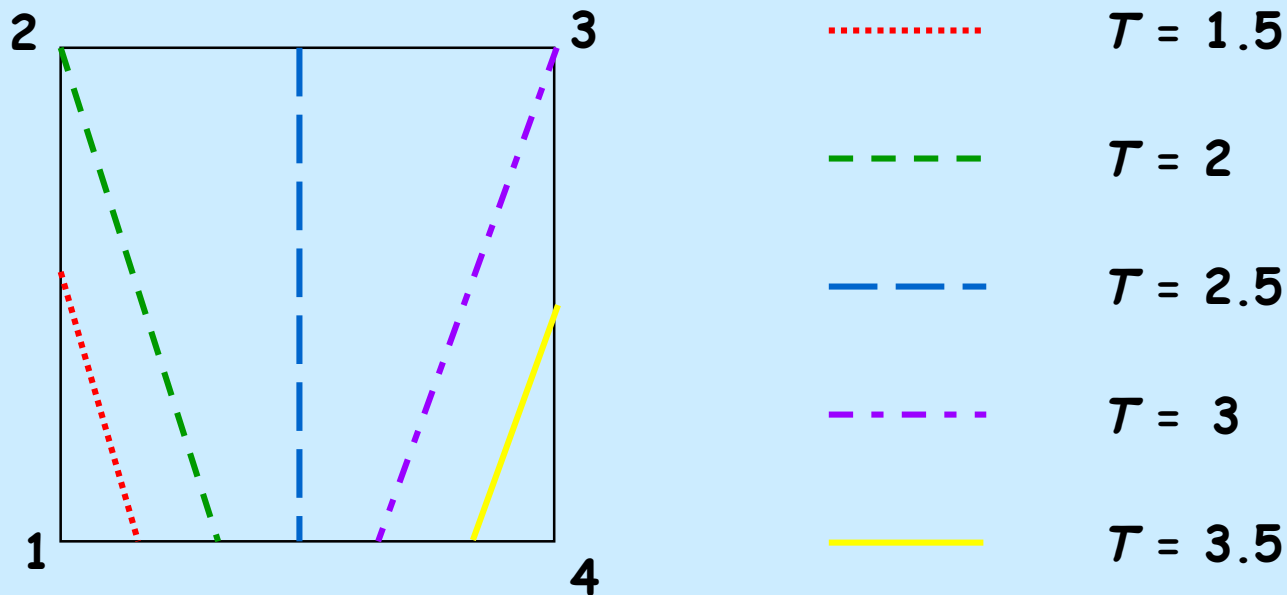
Volume Rendering: Marching Cubes

- *Marching cubes* is a technique for visualization of 3D data grids. For any given *threshold* or *level* T , it constructs the *isosurface* (level set) for T .
- The idea is easiest to explain by referring to a 2D data grid. The density values occur at the intersection points of the grid, and we consider the intersection of the isobar curves with each square of the grid independently.



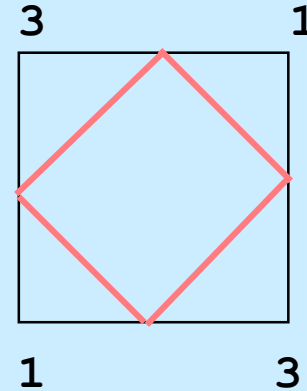
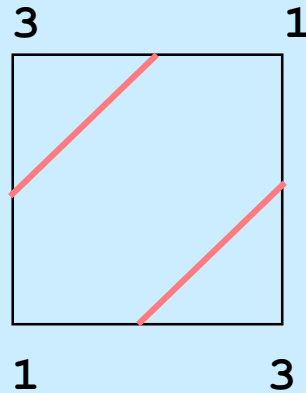
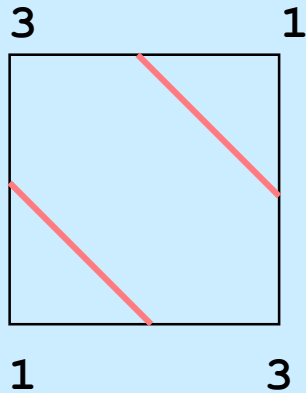
Volume Rendering: Marching Cubes

- To find the intersection of the isobar curves with a edges of the grid:
 - To determine topology, determine which edges the isobar intersects. For each such edge, the current threshold value \mathcal{T} lies between the density values stored at the edge's endpoints.
 - To determine geometry, for each edge that the isobar intersects, use linear interpolation to fix the point of intersection.



Volume Rendering: Marching Cubes

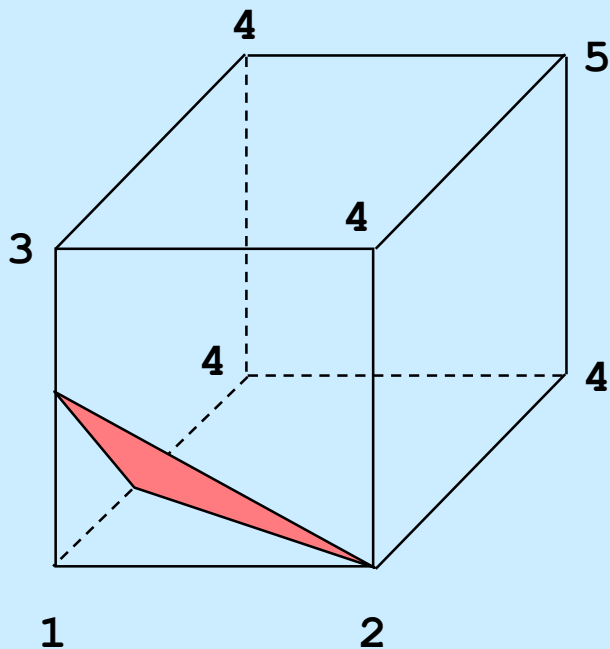
- Ambiguity occurs in some cases, due to under-sampling (i.e., use of a low-resolution data grid).



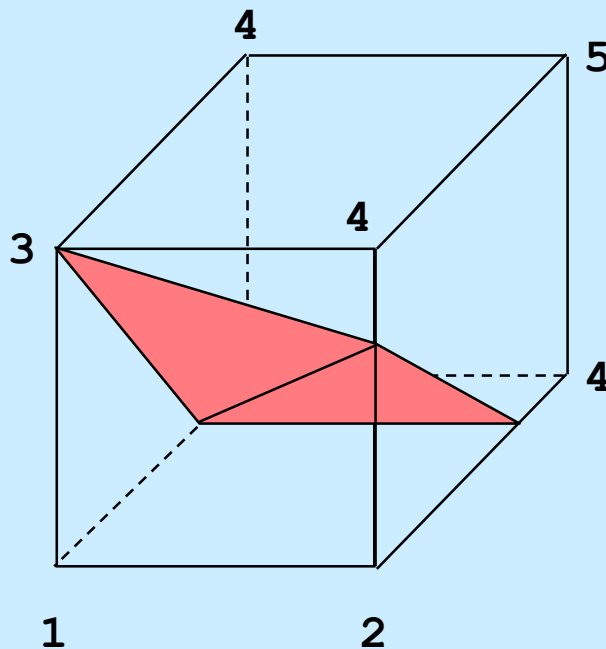
For $T = 2$, which of these topologies is right? Because the data is of limited resolution, there is no correct answer. However, it is sometimes possible to look at the density values of neighboring squares to decide which topology is most likely.

Volume Rendering: Marching Cubes

- In the 3D case, squares become cubes and isobars become isosurfaces. In the 3D data grid, we compute the intersection of the isosurface(s) for each cube independently:
 - To determine topology, decide which edges the isosurfaces cross.
 - To determine geometry, fix these crossing points by linear interpolation.



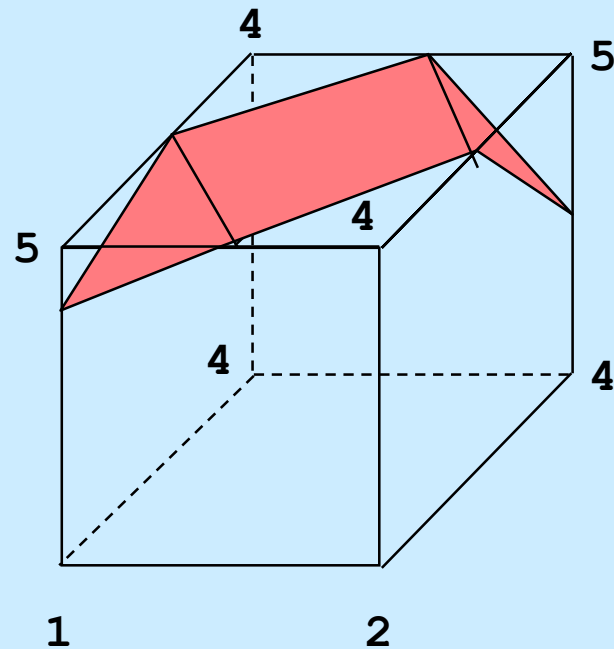
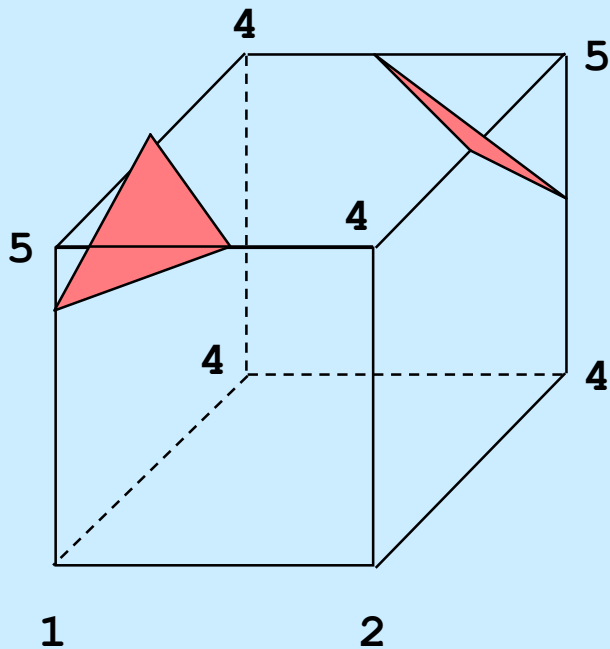
$T = 2$



$T = 3$

Volume Rendering: Marching Cubes

- Ambiguity can of course occur in 3D data grids. Below are 2 possible topologies for $T = 4.5$. For speed, the same topology is generally used in implementations; global features of the grid are usually not considered to resolve the ambiguity.



$T = 4.5$

Volume Rendering: Marching Cubes

- To implement marching cubes, the vertices of a generic cube are labeled with the integers 0 through 7. To compute the topology of the isosurface in a given cube, the cube's vertices are classified with respect to the threshold τ . Each vertex contributes:
 - 0 if the vertex's density value is less than τ , or
 - 1 if the vertex's density value is greater than τ .
- The 0s and 1s are concatenated into an 8-bit string in vertex-label order, thereby yielding an integer in the range [0..255]. This integer serves as an index into a topology look-up table.

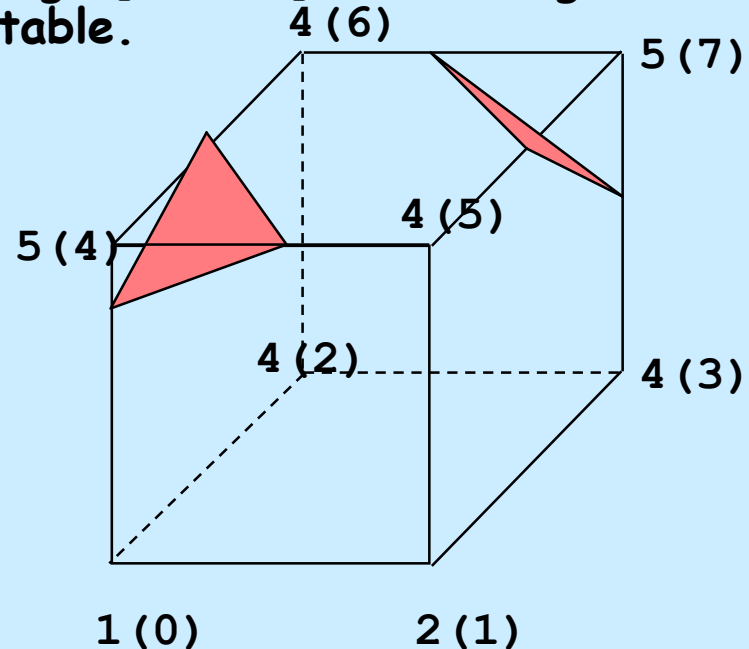
Here $a(b) = \text{densityValue}(\text{vertexLabel})$.

For $\tau = 4.5$, the index value is

1 0 0 1 0 0 0 0 (index)

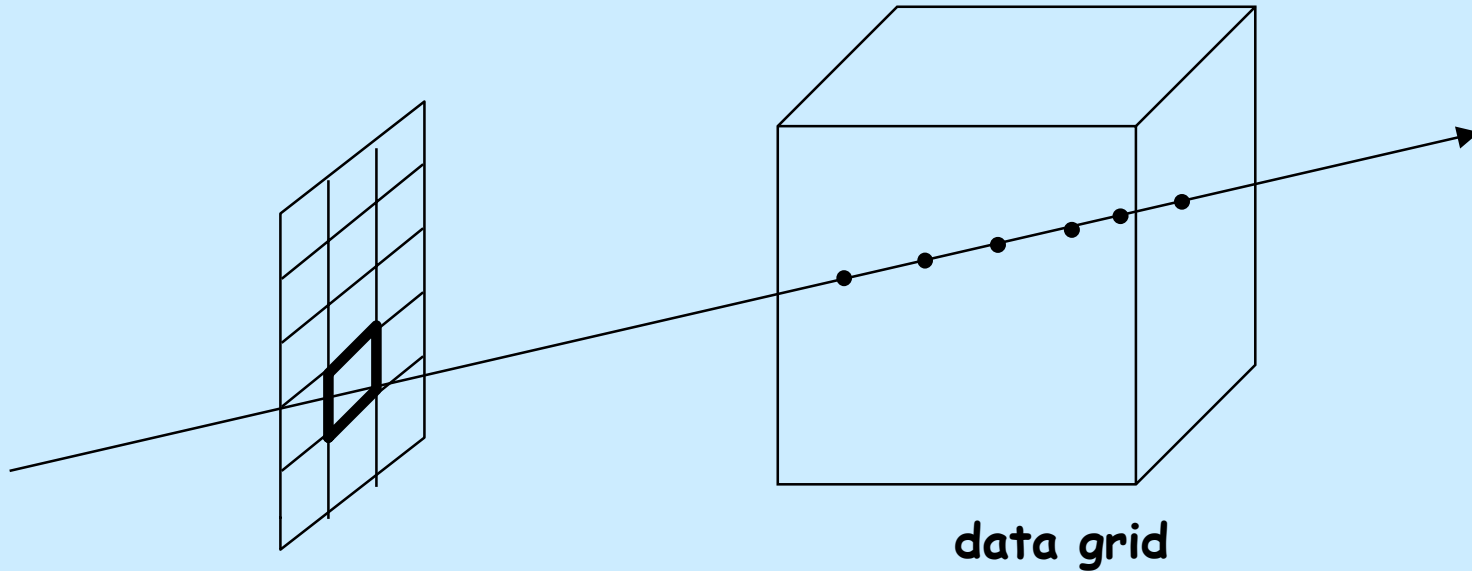
7 6 5 4 3 2 1 0 (bits)

yielding an index of 10010000 (base 2)
equal to 144 (base 10).

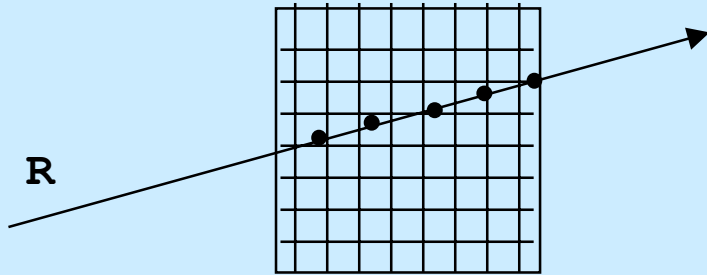


Volume Rendering: Volume Ray Tracing

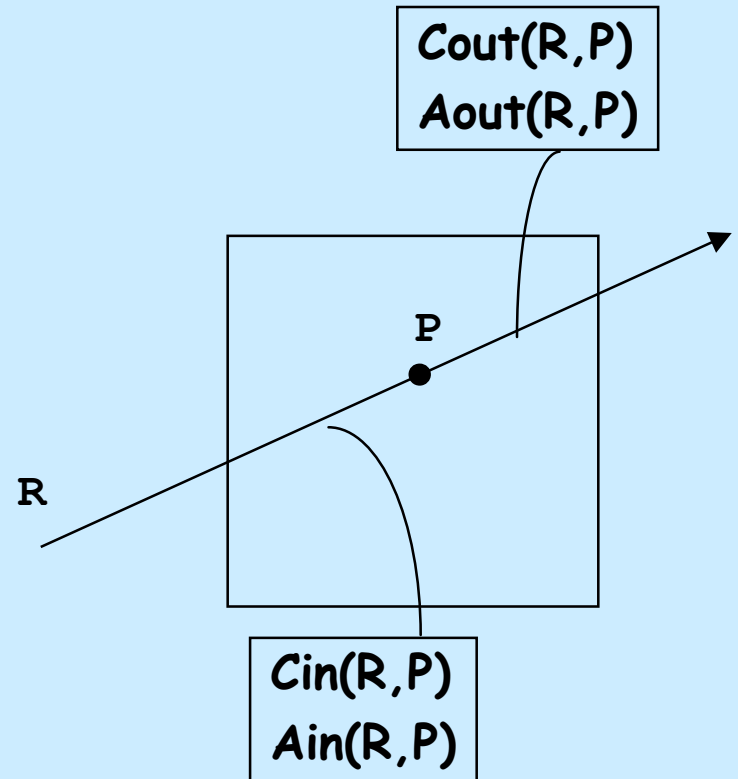
- To ray trace 3D data grids, we cast rays through the data grid while sampling density values at regular intervals. The density value at a point in a cube is computed by an interpolation scheme such as trilinear interpolation.
- A color transfer function C maps density values to RGB-color values.
- An opacity transfer function A maps density values to an opacity value in the range $[0,1]$, where 0 is transparent and 1 is opaque.



Volume Rendering: Volume Ray Tracing



Volume ray tracing in 2D



$C_{in}(R,P)$ = color of ray R upon entering sample point P .

$A_{in}(R,P)$ = opacity of ray R upon entering sample point P .

$C_{out}(R,P)$ = color of ray R upon leaving sample point P .

$A_{out}(R,P)$ = opacity of ray R upon leaving sample point P .

Volume Rendering: Volume Ray Tracing

$C_{in}(R,P)$ = color of ray R upon entering sample point P.

$A_{in}(R,P)$ = opacity of ray R upon entering sample point P.

$C_{out}(R,P)$ = color of ray R upon leaving sample point P.

$A_{out}(R,P)$ = opacity of ray R upon leaving sample point P.

$C(P)$ = color of P (determined by color transfer function).

$A(P)$ = opacity of P (based on opacity transfer function).

Given $C_{in}(R,P)$, $A_{in}(R,P)$, $C(P)$, and $A(P)$, we can compute $C_{out}(R,P)$ and $A_{out}(R,P)$:

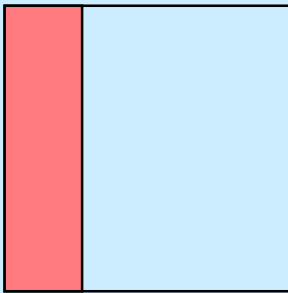
- $C_{out}(R,P) = C_{in}(R,P) * A_{in}(R,P) + C(P) * (1 - A_{in}(R,P))$
- $A_{out}(R,P) = A_{in}(R,P) + A(P) * (1 - A_{in}(R,P))$

Volume Rendering: Volume Ray Tracing

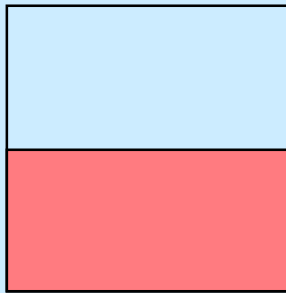
- The relation:

$$A_{out}(R,P) = A_{in}(R,P) + A(P) * (1 - A_{in}(R,P))$$

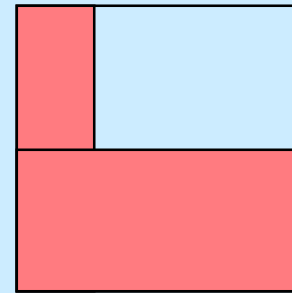
can be understood using a metaphor of area coverage:



$$A_{in}(R,P) = 1/4$$



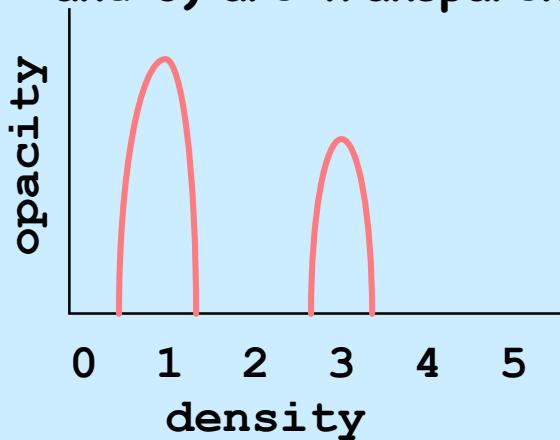
$$A(P) = 1/2$$



$$\begin{aligned} A_{out}(R,P) &= A_{in}(R,P) + A(P) * (1 - A_{in}(R,P)) \\ &= 1/4 + (1/2) * (3/4) \\ &= 5/8 \end{aligned}$$

Volume Rendering: Volume Ray Tracing

- To detect isosurfaces while ray tracing volumes, we define the color transfer and opacity transfer function such that they are high for the selected level values and low otherwise. For instance, the following functions paint the isosurface for $L=1$ the color red, and the isosurface for $L=3$ the color green. All other density (outside neighborhoods of 1 and 3) are transparent to the rays.



opacity transfer
function A

