What is an iso-contour?

What is an iso-contour?

A set of points in the data that have the same scalar value

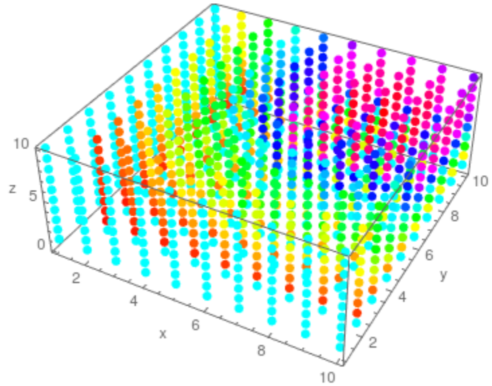
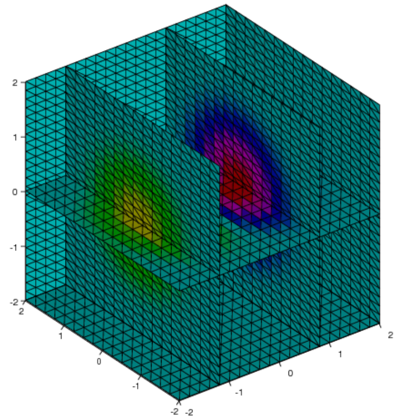
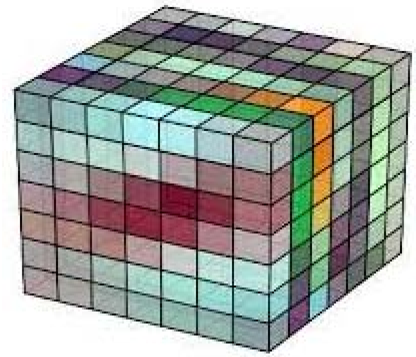
What are the advantages of iso-contouring?

What is an iso-contour?

A set of points in the data that have the same scalar value

What are the advantages of iso-contouring?

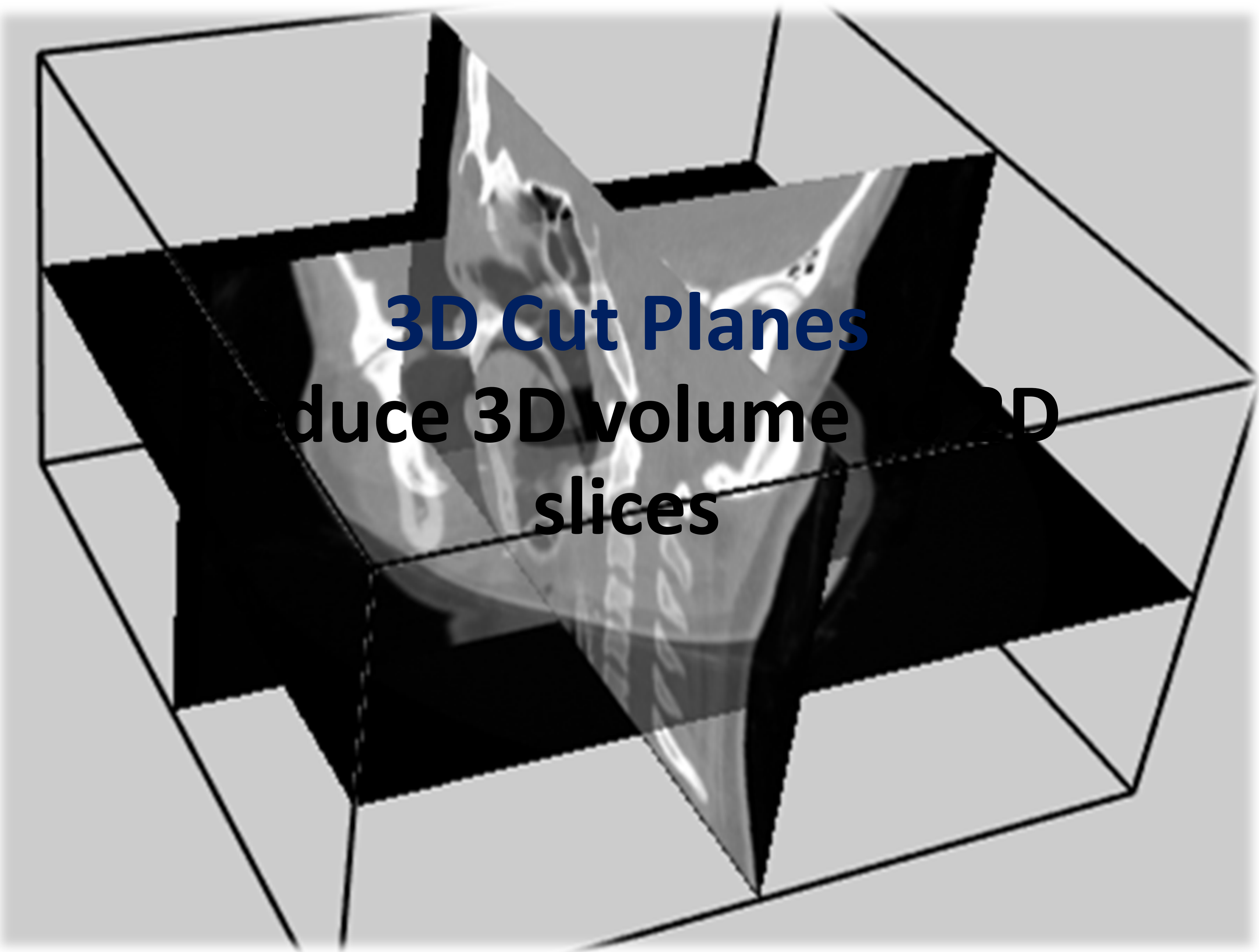
provide more detailed and precise depiction of the patterns in 2D scalar fields.

Scalar Field Visualization – 3D

Cutting Planes & Iso-surfacing

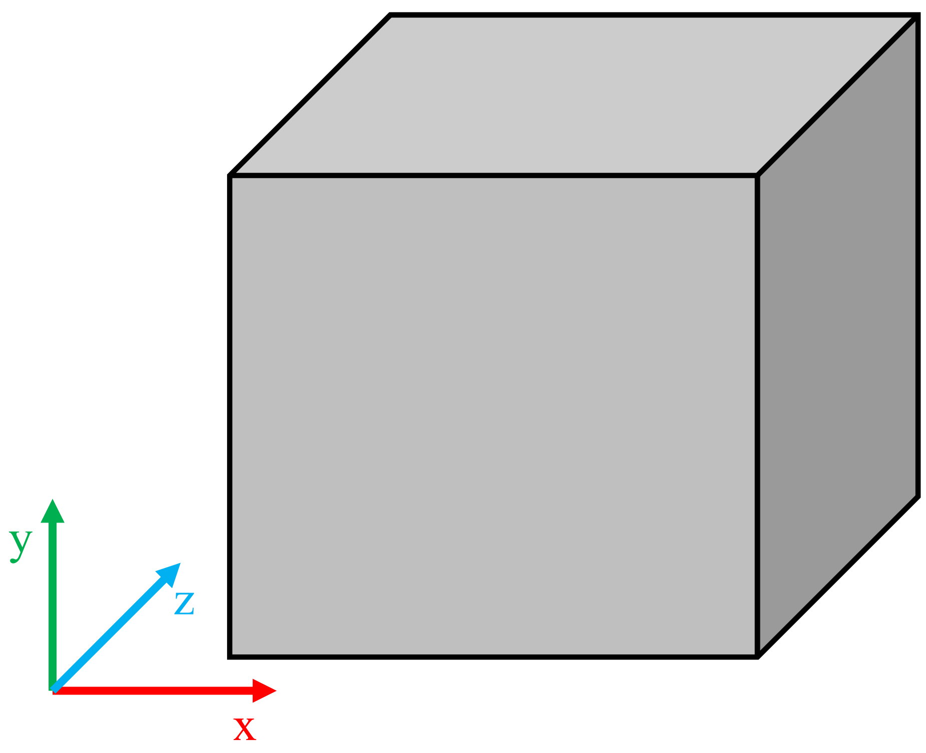
Goal: know the simple cutting plane based visualization and the construction of iso-surfacing



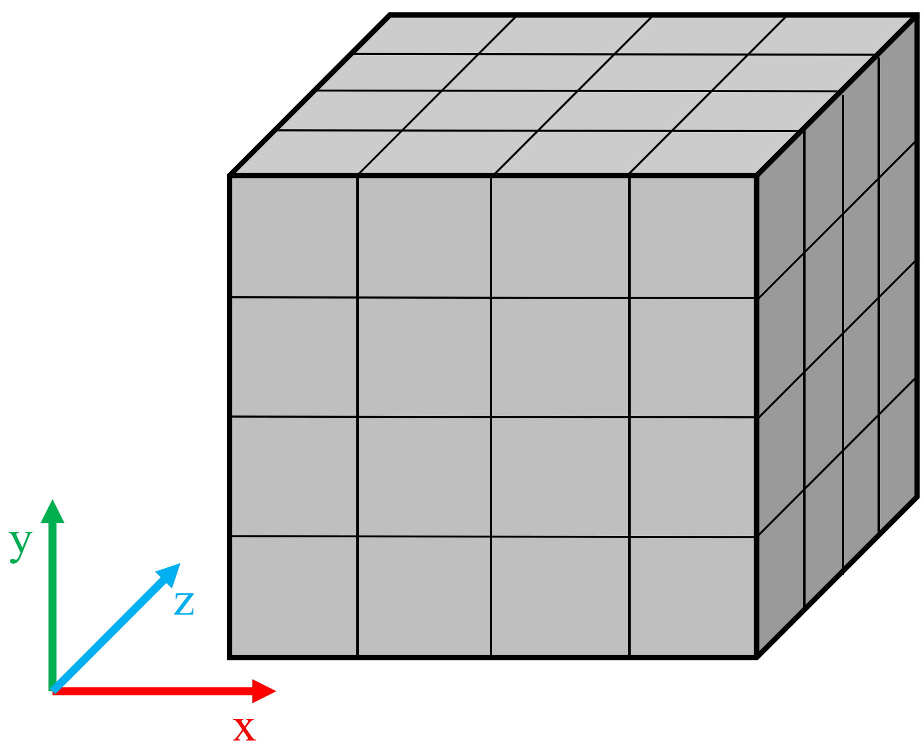
3D Cut Planes

Reduce 3D volume to 2D slices

Here, we focus on axis-aligned cut planes, assuming the 3D volume is also axis-aligned

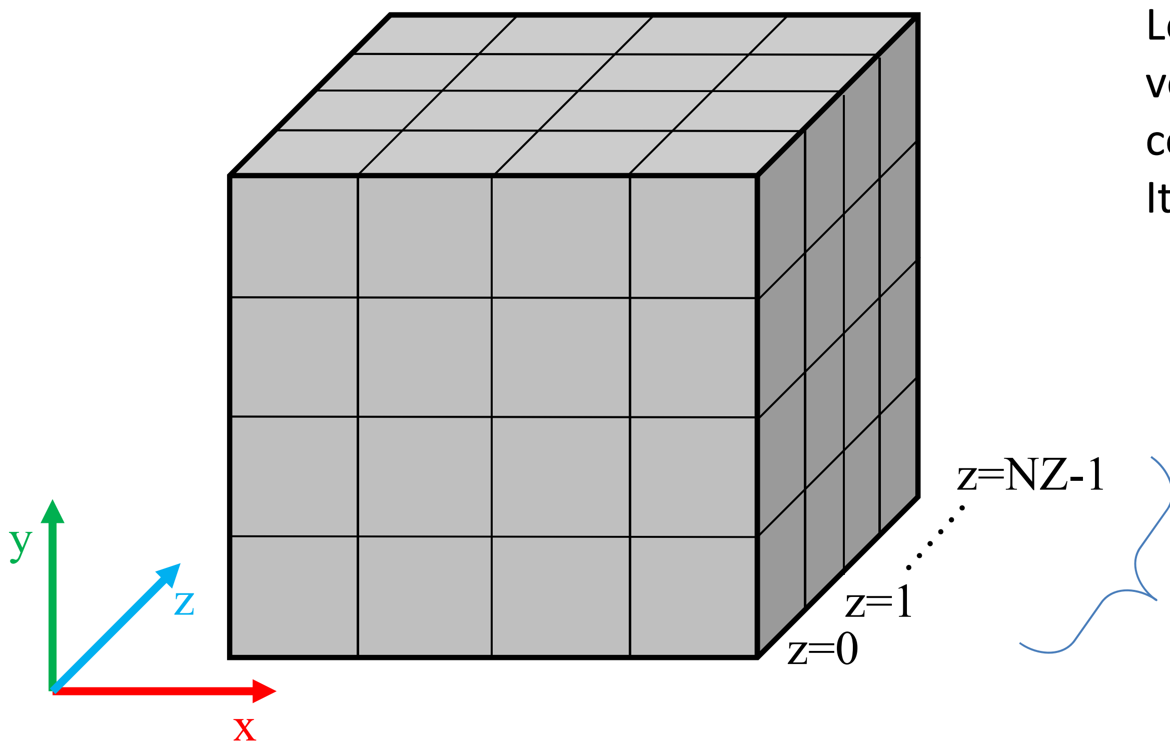


Here, we focus on axis-aligned cut planes, assuming the 3D volume is also axis-aligned



Let us concern with a uniform volume (or 3D image), which consists of individual **voxels**. Its dimension is NX*NY*NZ.

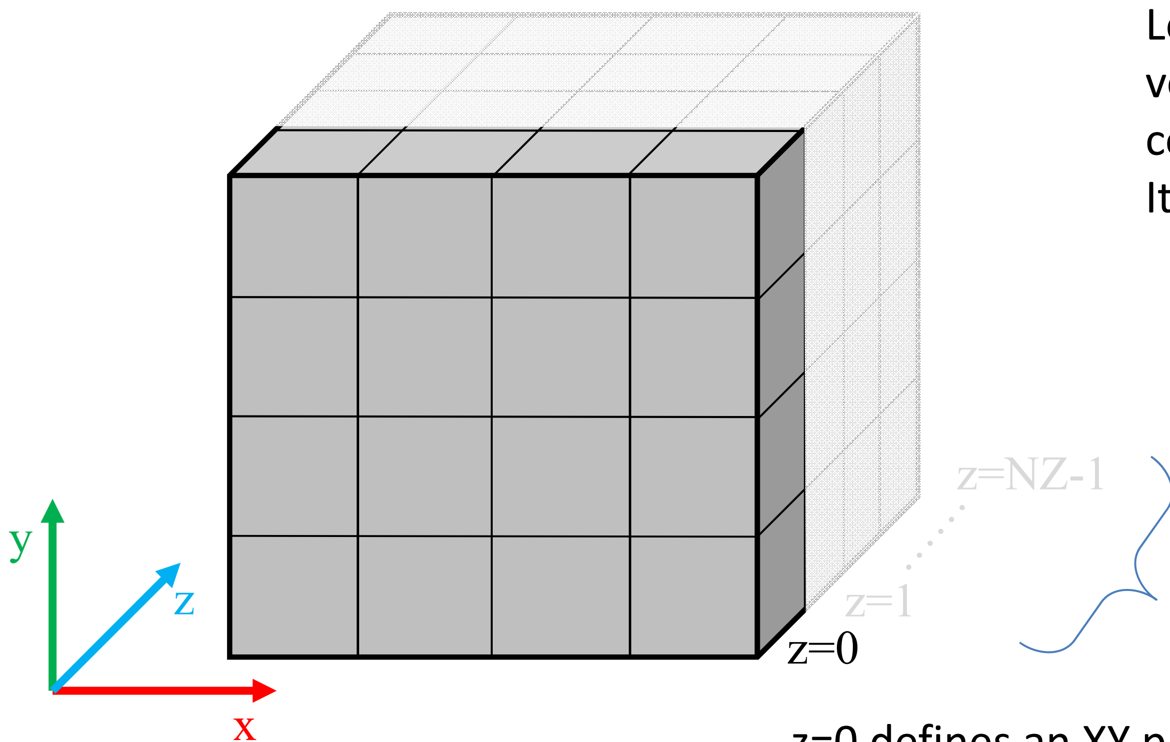
Here, we focus on axis-aligned cut planes, assuming the 3D volume is also axis-aligned



Let us concern with a uniform volume (or 3D image), which consists of individual voxels. Its dimension is NX*NY*NZ.

Now let us look at z dimension

Here, we focus on axis-aligned cut planes, assuming the 3D volume is also axis-aligned



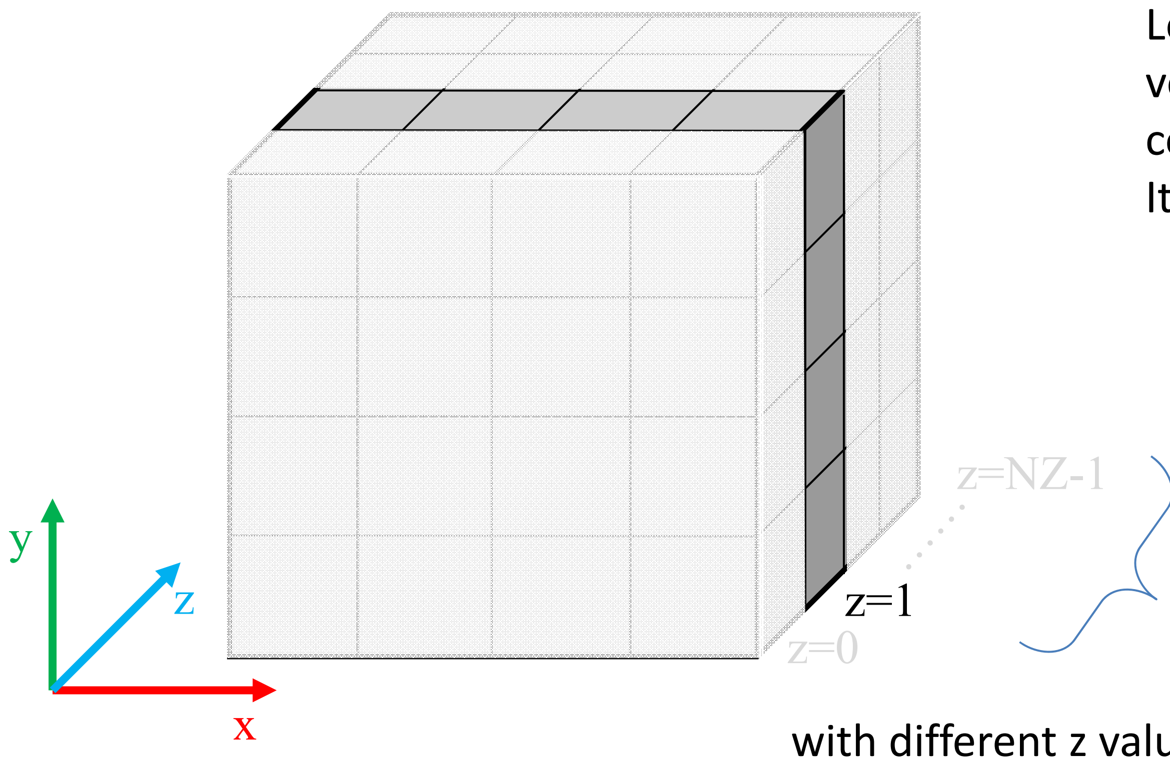
Let us concern with a uniform volume (or 3D image), which consists of individual voxels. Its dimension is NX*NY*NZ.

Now let us look at z dimension

z=0 defines an XY planes with all voxels whose z index is zero.

Here, we focus on axis-aligned cut planes, assuming the 3D volume is also axis-aligned

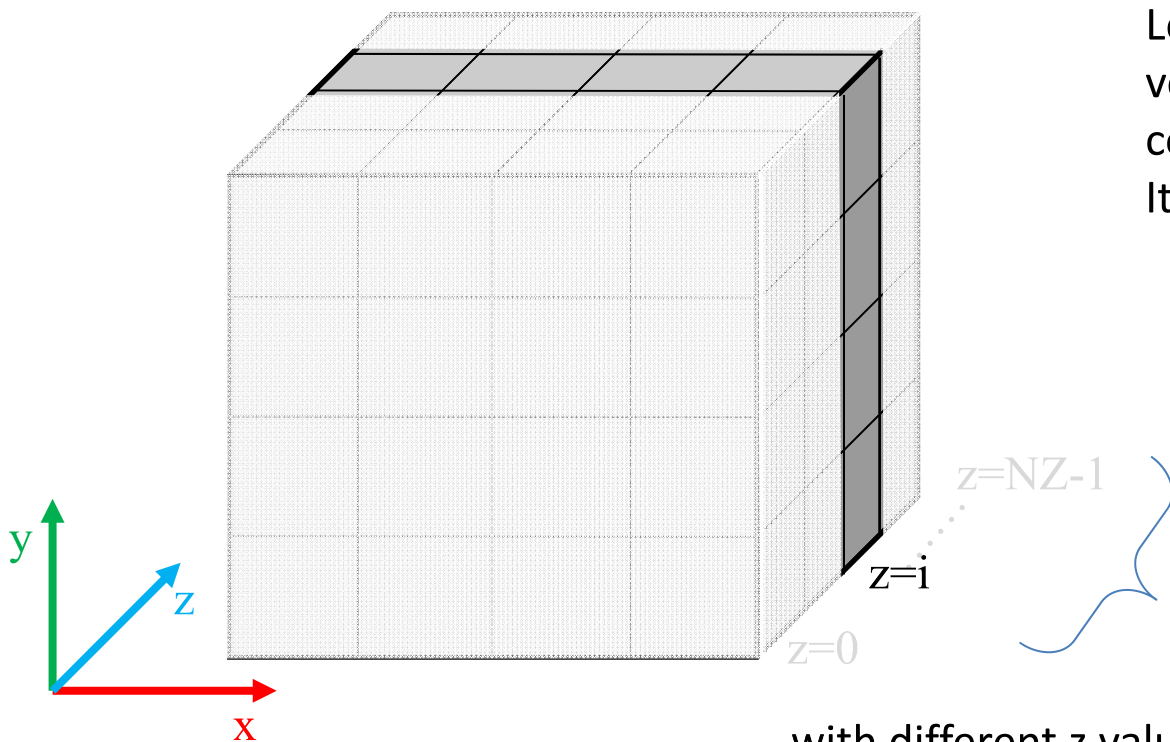
Let us concern with a uniform volume (or 3D image), which consists of individual voxels. Its dimension is NX*NY*NZ.



Now let us look at z dimension

with different z values (indices), you get different XY planes with different z indices

Here, we focus on axis-aligned cut planes, assuming the 3D volume is also axis-aligned

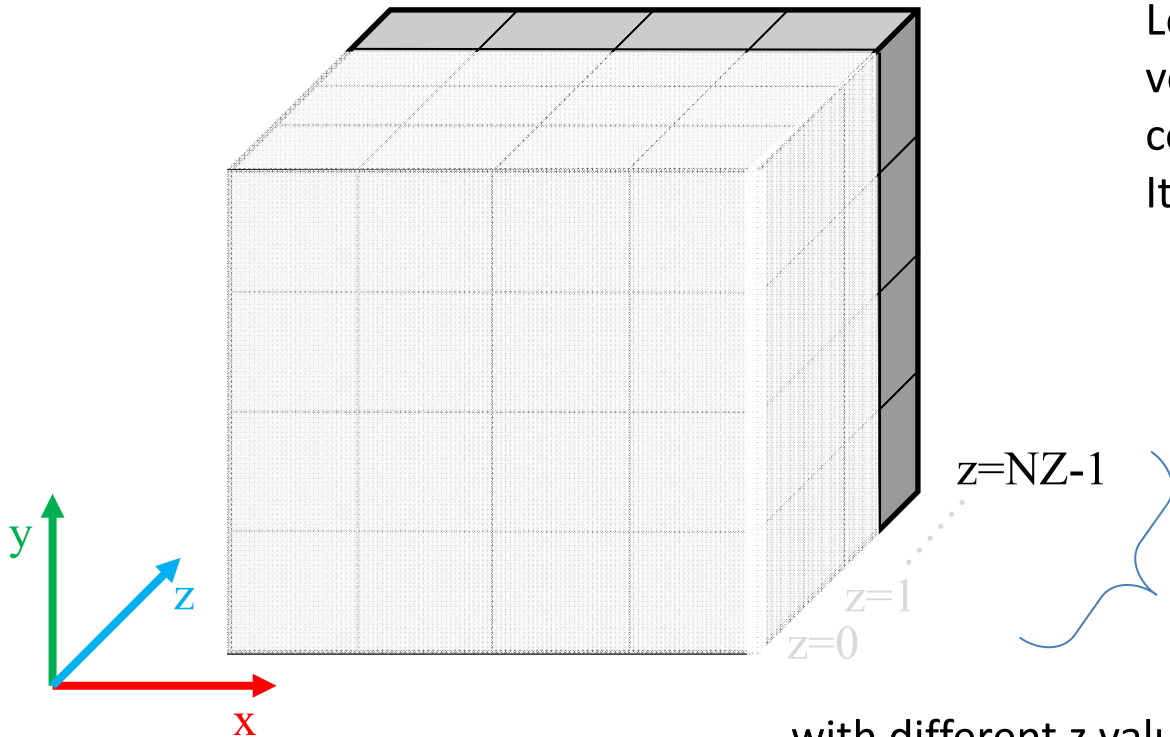


Let us concern with a uniform volume (or 3D image), which consists of individual voxels. Its dimension is NX*NY*NZ.

Now let us look at z dimension

with different z values (indices), you get different XY planes with different z indices

Here, we focus on axis-aligned cut planes, assuming the 3D volume is also axis-aligned

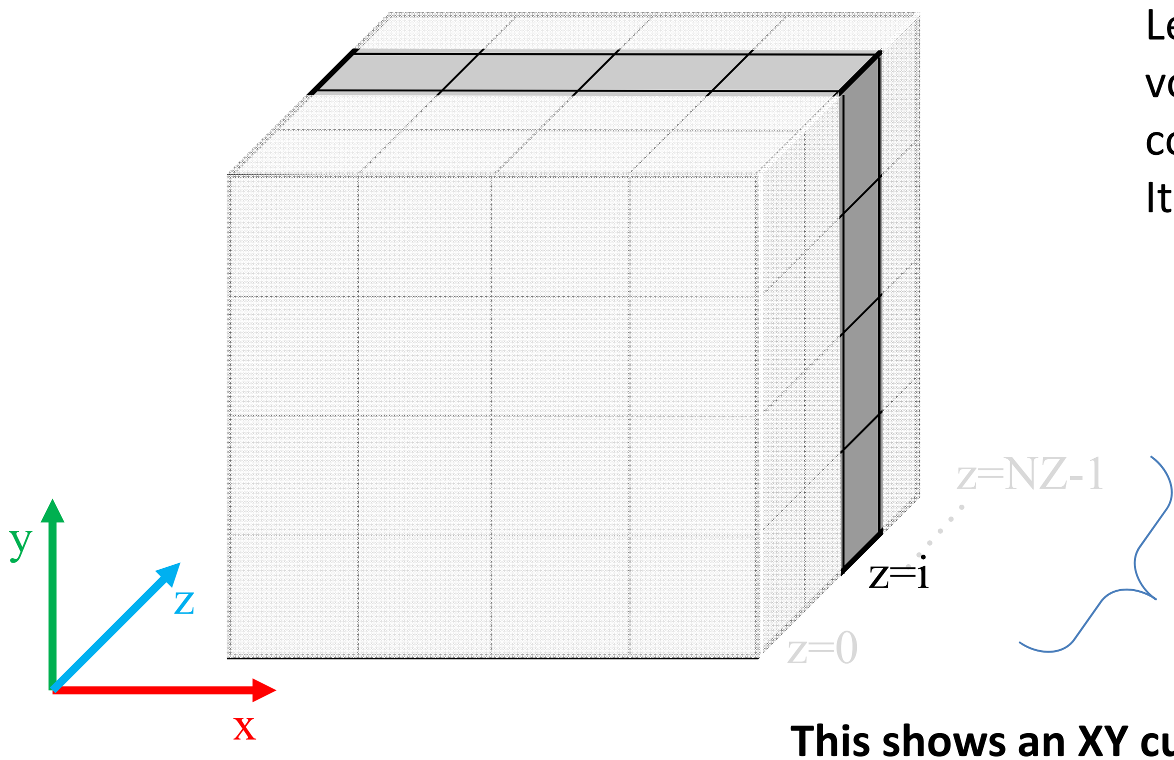


Let us concern with a uniform volume (or 3D image), which consists of individual voxels. Its dimension is NX*NY*NZ.

Now let us look at z dimension

with different z values (indices), you get different XY planes with different z indices

Here, we focus on axis-aligned cut planes, assuming the 3D volume is also axis-aligned



Let us concern with a uniform volume (or 3D image), which consists of individual voxels. Its dimension is NX*NY*NZ.

Now let us look at z dimension

This shows an XY cut plane (perpendicular to z) can be constructed by specifying a z value between [0, NZ-1].

In VTK

Use the following to get the dimension of the 3D image data or structured grid

```
dim = reader.GetOutput().GetDimensions()
```

```
# Create a mapper and assign it to the corresponding reader
xy_plane_Colors = vtk.vtkImageMapToColors()
xy_plane_Colors.SetInputConnection(reader.GetOutputPort())
xy_plane_Colors.SetLookupTable([you color look up table])
xy_plane_Colors.Update()
```

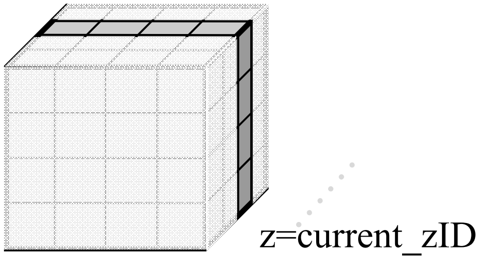
In VTK

Use the following to get the dimension of the 3D image data or structured grid

```
dim = reader.GetOutput().GetDimensions()
```

```
# Create a mapper and assign it to the corresponding reader
xy_plane_Colors = vtk.vtkImageMapToColors()
xy_plane_Colors.SetInputConnection(reader.GetOutputPort())
xy_plane_Colors.SetLookupTable([you color look up table])
xy_plane_Colors.Update()
```

```
# Create an image actor for the XY plane
xy_plane = vtk.vtkImageActor()
xy_plane.GetMapper().SetInputConnection(xy_plane_Colors.GetOutputPort())
xy_plane.SetDisplayExtent(0, dim[0]-1, 0, dim[1]-1, current_zID,
current_zID)
# Current_zID is a user-input integer within the range of [0, zdim-1]
```



In VTK

Use the following to get the dimension of the 3D image data or structured grid

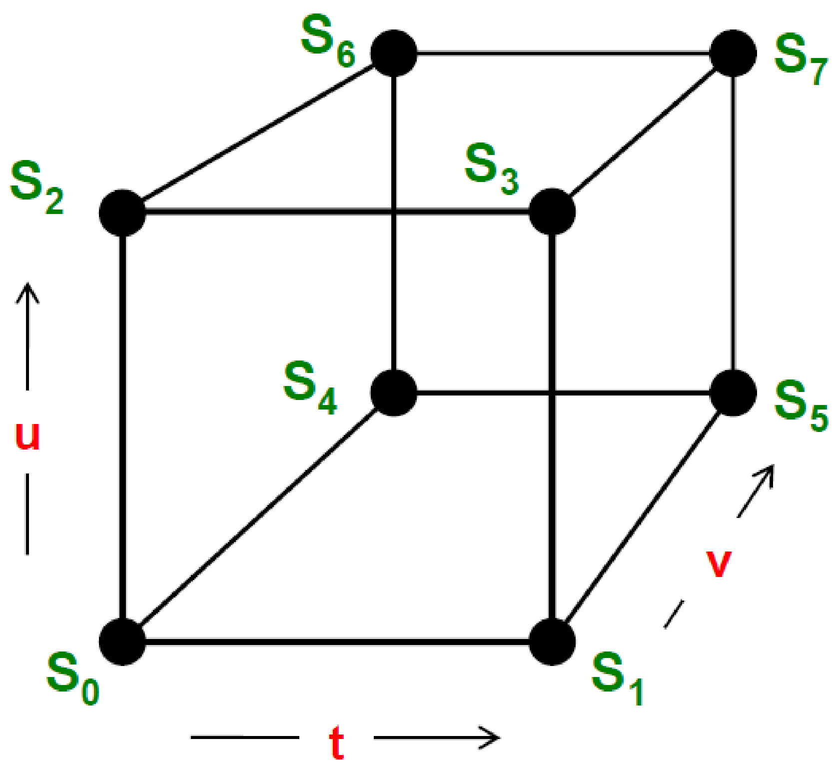
```
dim = reader.GetOutput().GetDimensions()
```

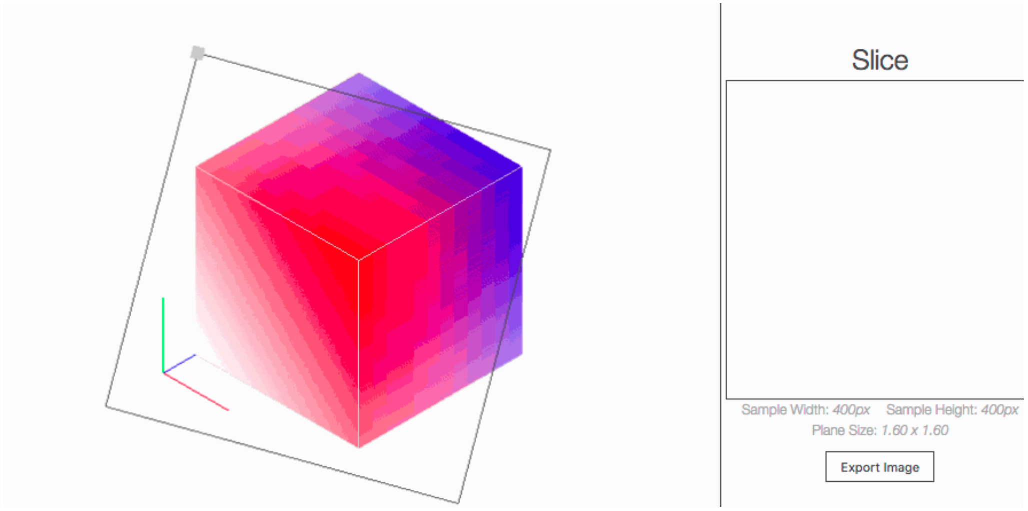
```
# Create a mapper and assign it to the corresponding reader
xy_plane_Colors = vtk.vtkImageMapToColors()
xy_plane_Colors.SetInputConnection(reader.GetOutputPort())
xy_plane_Colors.Update()

# Create an image actor for the XY plane
xy_plane = vtk.vtkImageActor()
xy_plane.GetMapper().SetInputConnection(xy_plane_Colors.GetOutputPort())
xy_plane.SetDisplayExtent(0, dim[0]-1, 0, dim[1]-1, current_zID, current_zID)
```

YZ and XZ cut planes can be similarly added!!!

This is a task of your assignment 3.
You also need to play with the ***transfer function*** *for the color plots shown in the individual cut planes*

Trilinear Interpolation



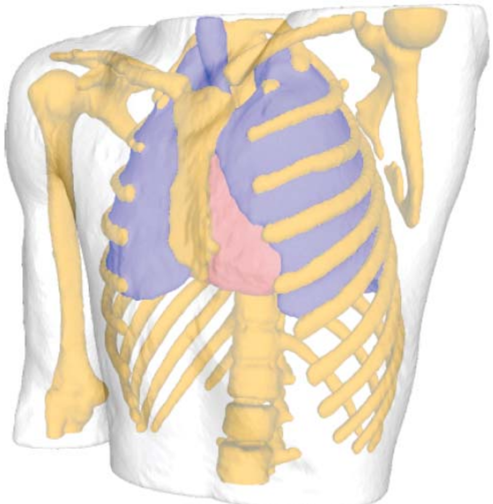


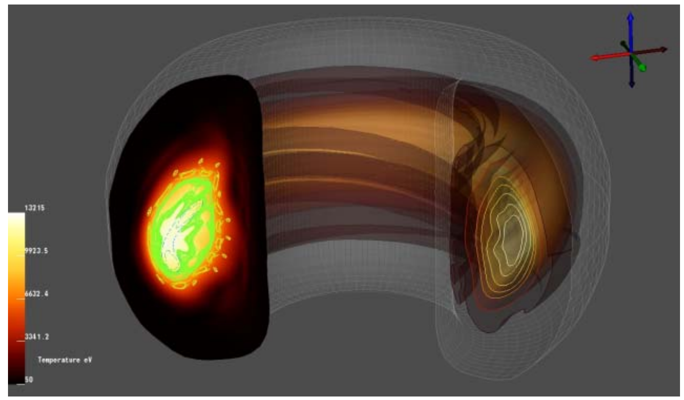
$$S(t,u,v) = (1-t)(1-u)(1-v)S_0 + t(1-u)(1-v)S_1 + (1-t)u(1-v)S_2 + tu(1-v)S_3 + (1-t)(1-u)vS_4 + t(1-u)vS_5 + (1-t)uvS_6 + tuvS_7$$

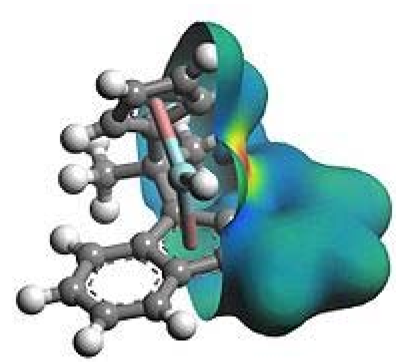
This is useful, for example, if we have passed an oblique cutting plane through a 3D mesh of points and are trying to interpolate scalar values from the 3D mesh to the 2D plane.

Iso-surfacing

Iso-Surfaces: Applications







A contour line is often called an ***iso-line***, that is a line/curve of equal value. When hiking, for example, if you could walk along a single contour line of the terrain, you would remain at the same elevation.

An iso-surface is the same idea, only in 3D. It is a surface of equal value.

Sometimes the shapes of the iso-surfaces **have a physical meaning**, such as bone, skin, different layers of earth etc. (e.g., the left example above). Sometimes the shape just **helps turn an abstract notion into something physical** to help us gain insight (e.g., the other two examples).

Iso-surface Construction: Marching Cubes

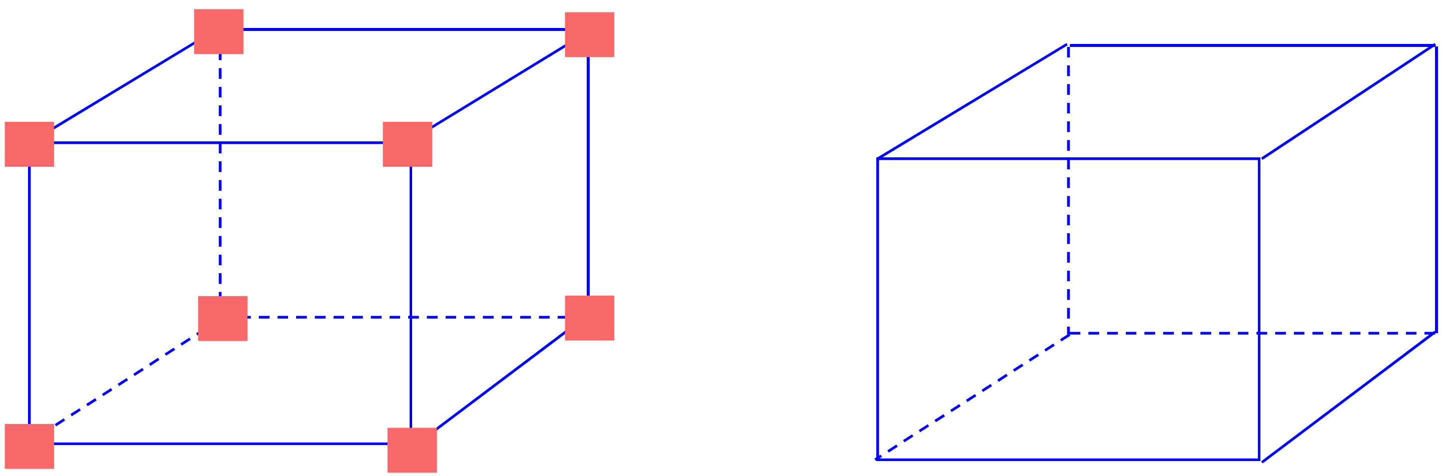
Similar to Marching Squares, we go through individual cubes to construct a patch of the iso-surface

- For simplicity, we shall work with zero level ($s^*=0$) iso-surface, and denote

positive vertices as

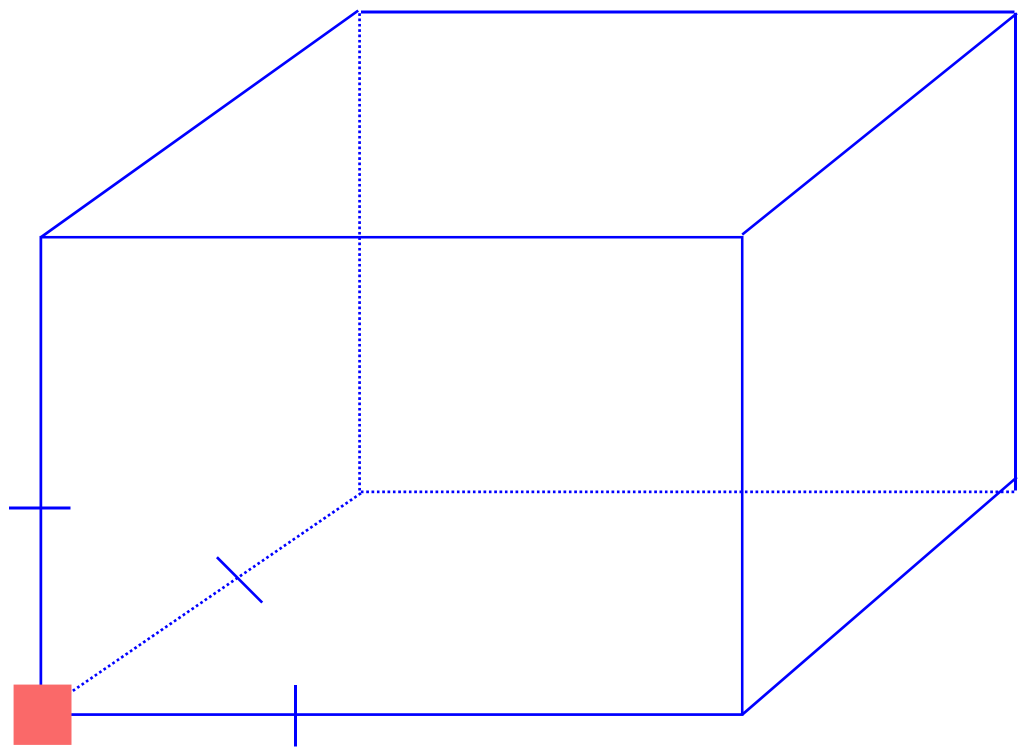
There are **EIGHT** vertices, each can be positive or negative - so there are **2^8 = 256** different cases!

These two are easy!



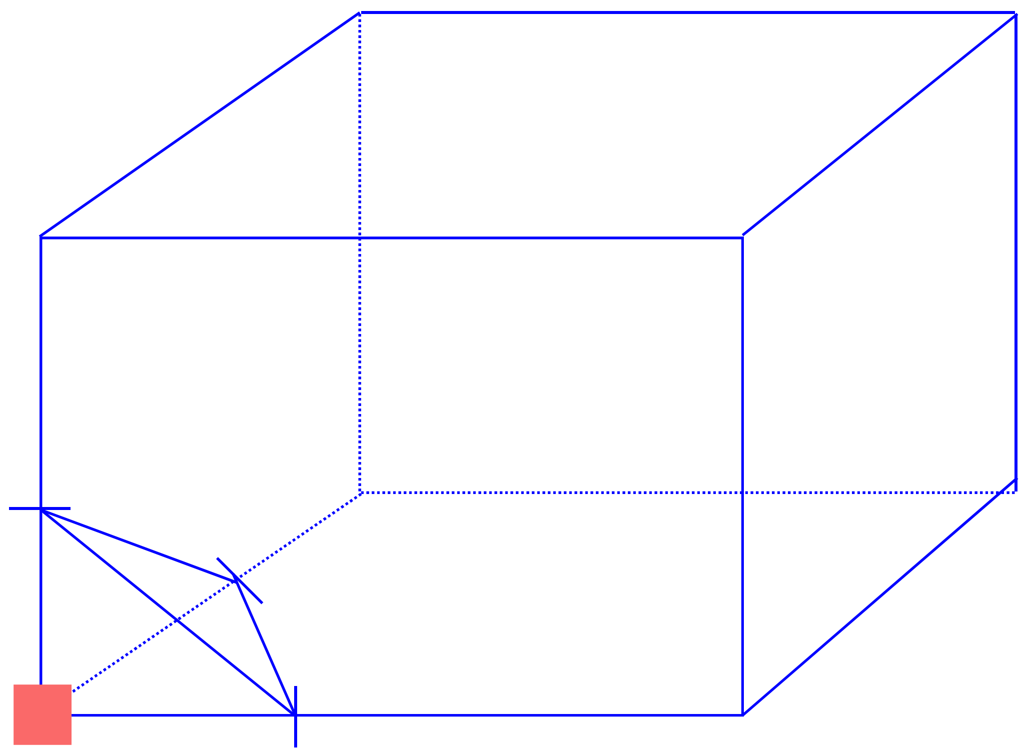
There is no portion of the iso-surface inside the cube!

Iso-surface Construction - One Positive Vertex - 1



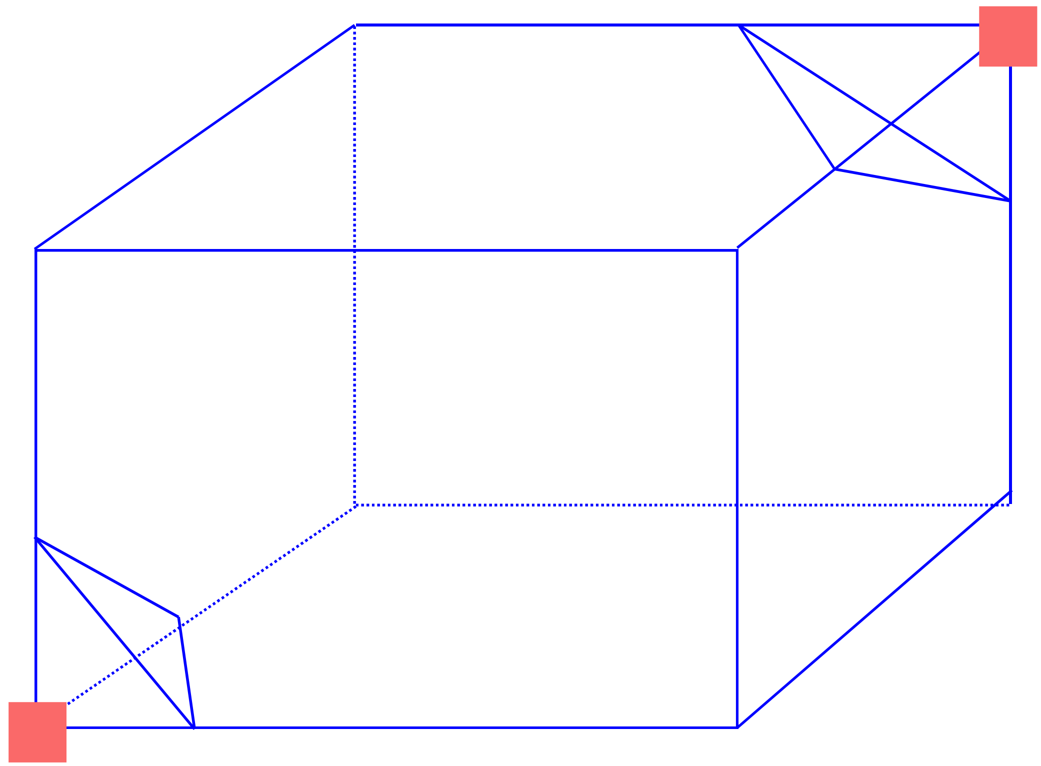
Intersections with edges found by inverse linear interpolation (as in iso-contouring)

Iso-surface Construction - One Positive Vertex - 2



Joining edge intersections across faces forms a triangle as part of the iso-surface

Isosurface Construction -Positive Vertices at Opposite Corners



Iso-surface Construction: Marching Cubes

- One can work through all 256 cases in this way - although it quickly becomes apparent that many cases are similar.

Iso-surface Construction: Marching Cubes

- One can work through all 256 cases in this way - although it quickly becomes apparent that many cases are similar.

- For example:
  - 2 cases where all are positive, or all negative, give no iso-surface
  - 16 cases where one vertex has opposite sign from all the rest

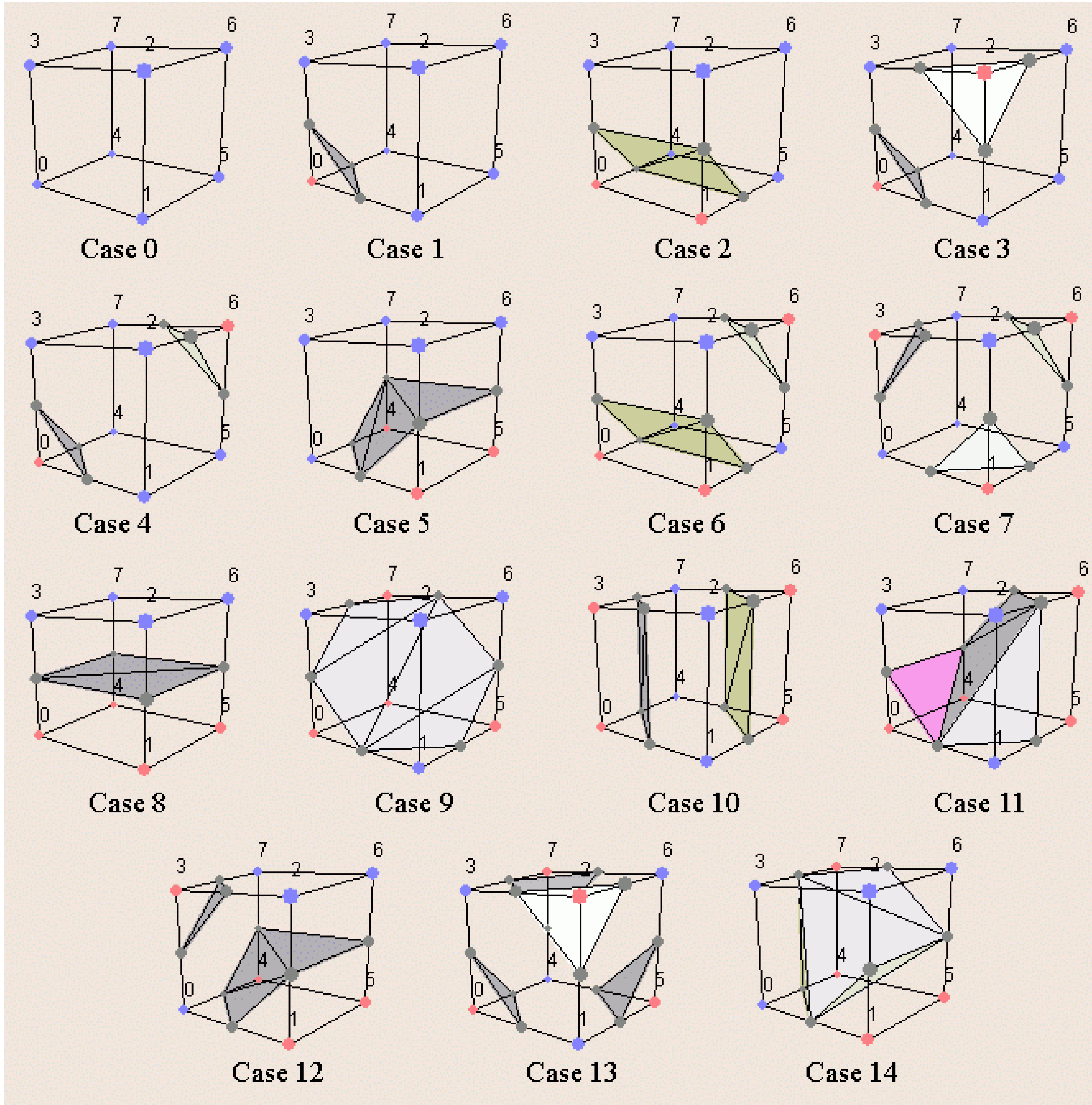
Iso-surface Construction: Marching Cubes

- One can work through all 256 cases in this way - although it quickly becomes apparent that many cases are similar.

- For example:
  - 2 cases where all are positive, or all negative, give no isosurface
  - 16 cases where one vertex has opposite sign from all the rest

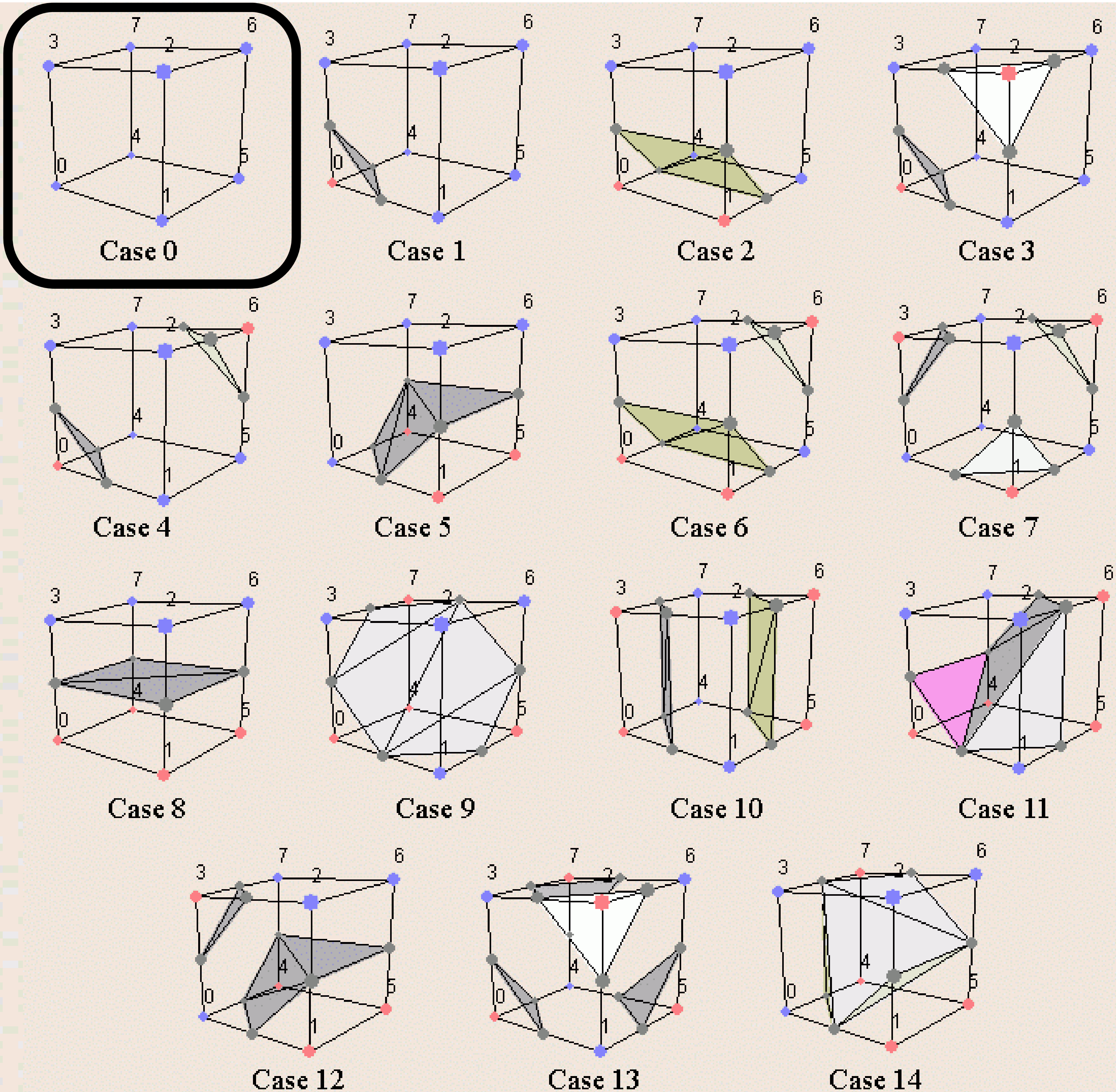
- In fact, there are **only 15** topologically distinct configurations

Case Table


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
Case 14

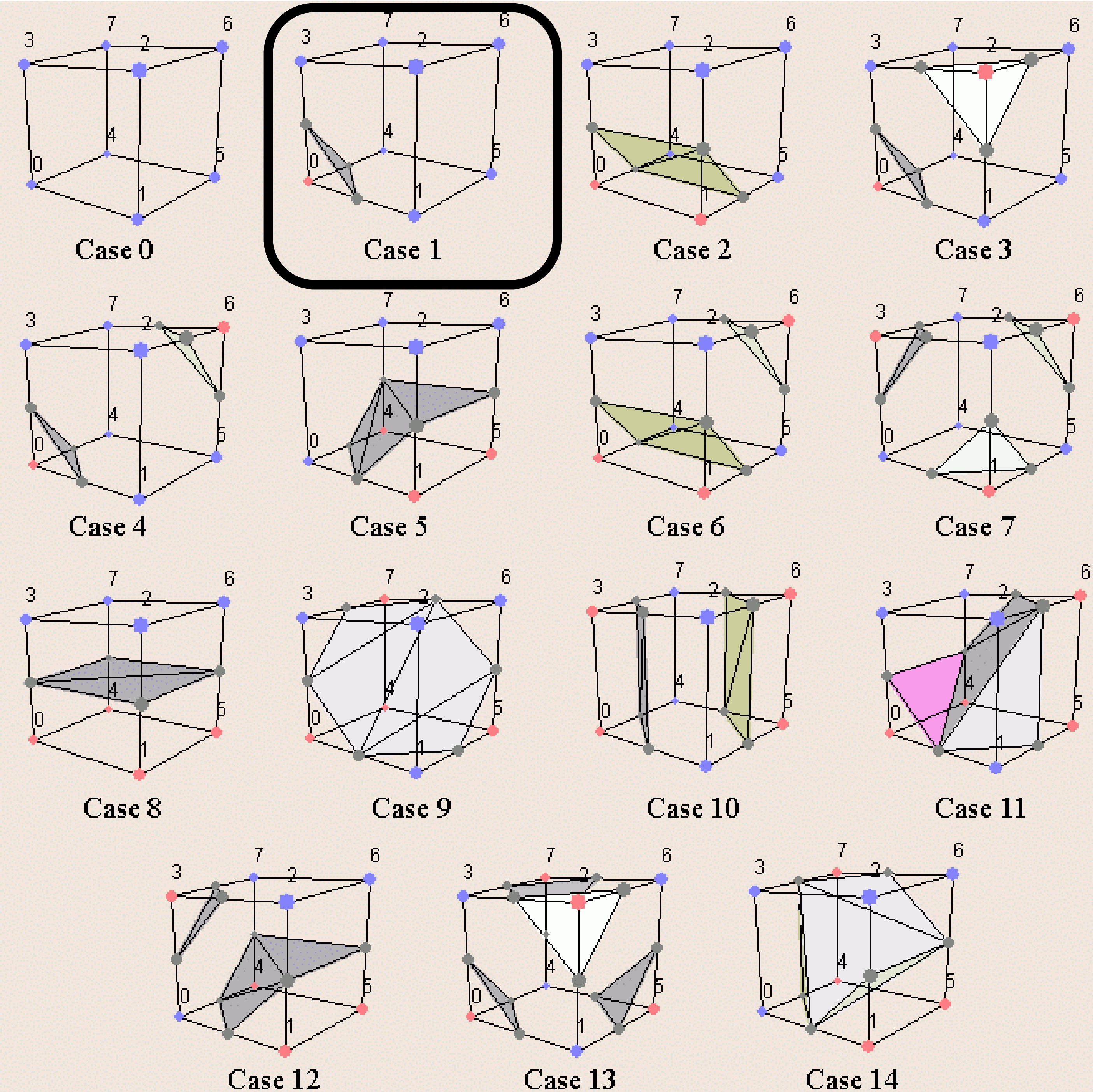
● 8 Above
○ 0 Below

1 case



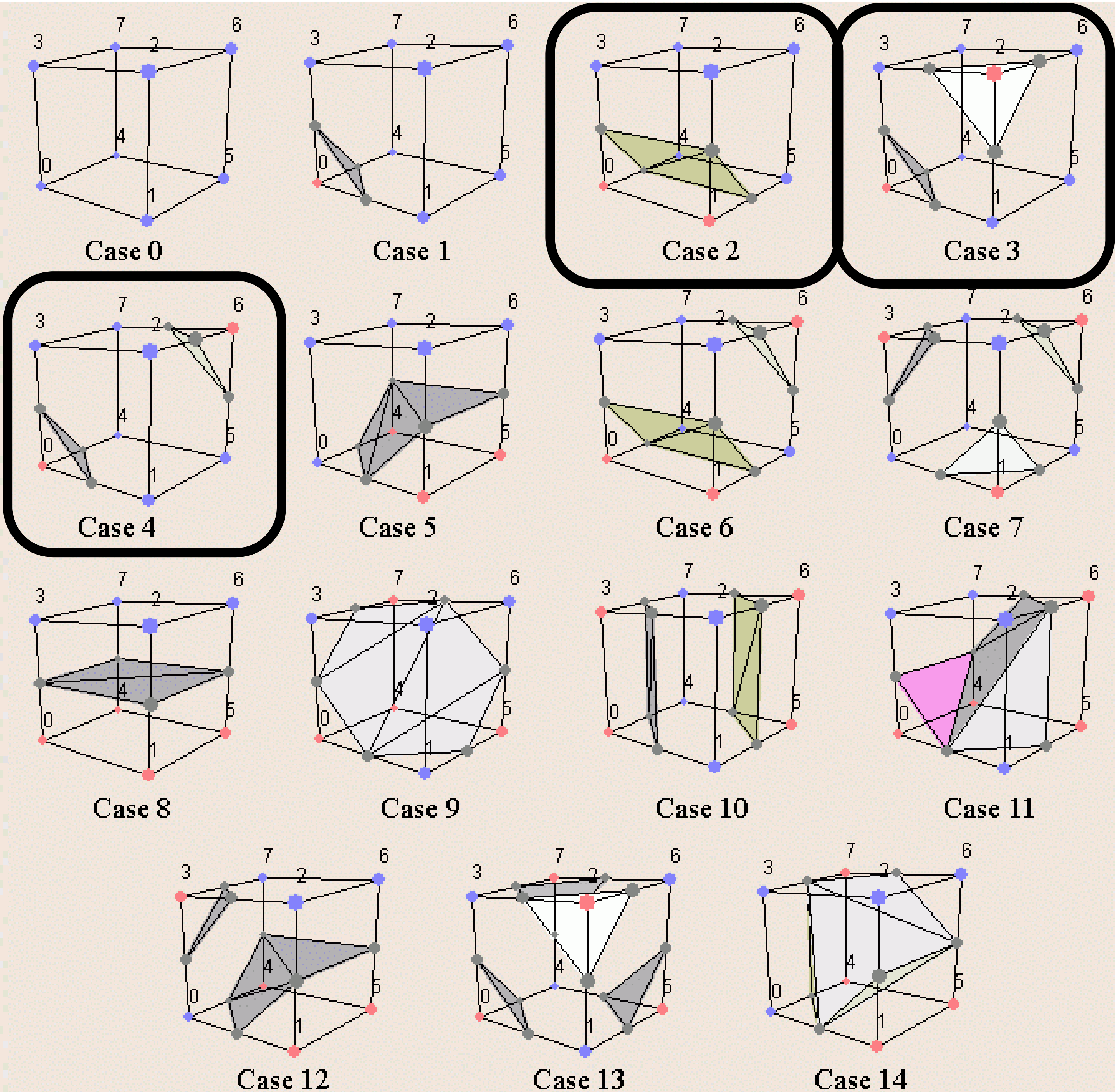
- 7 Above
- 1 Below

1 case



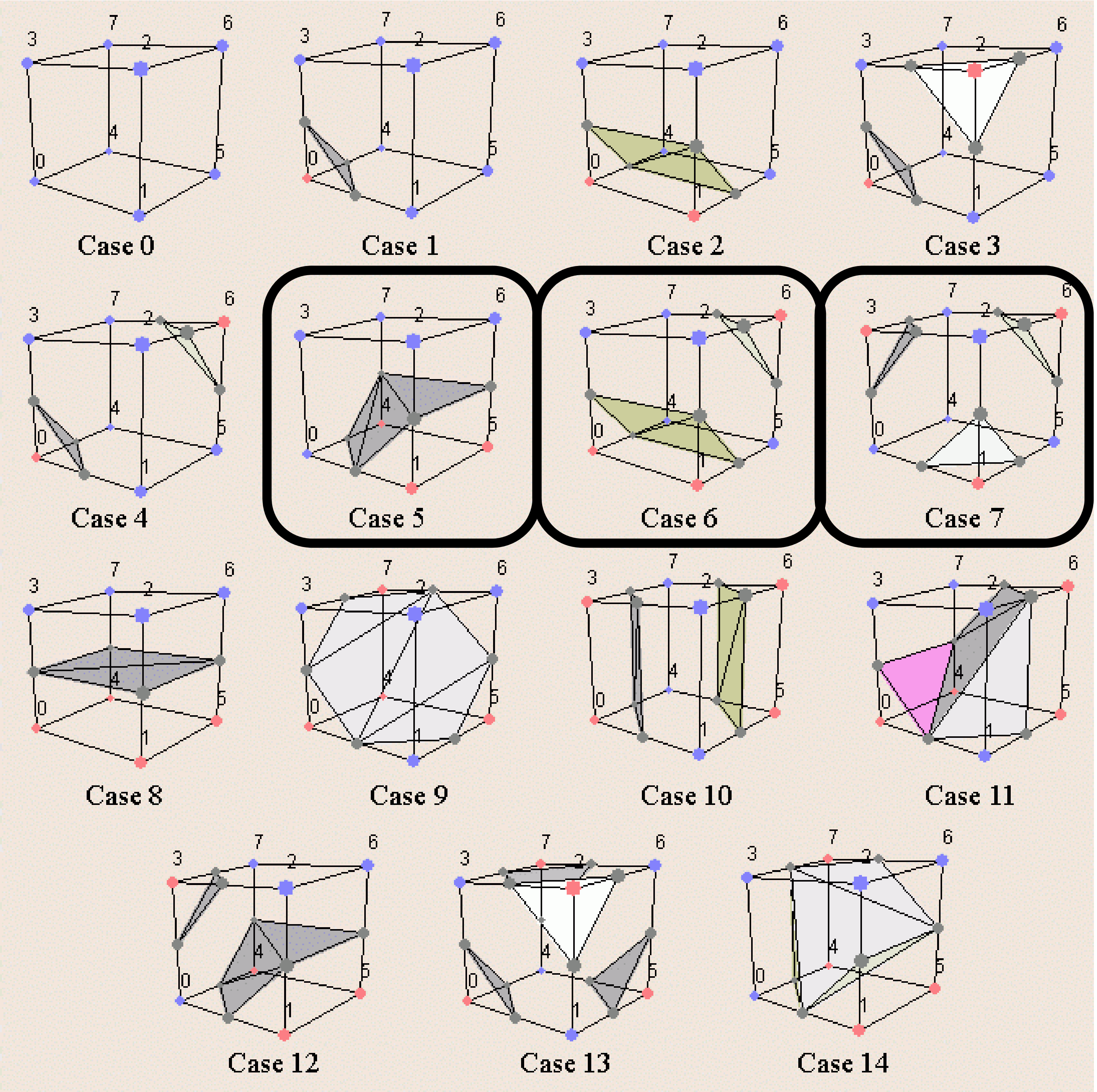
- 6 Above
- 2 Below

3 cases



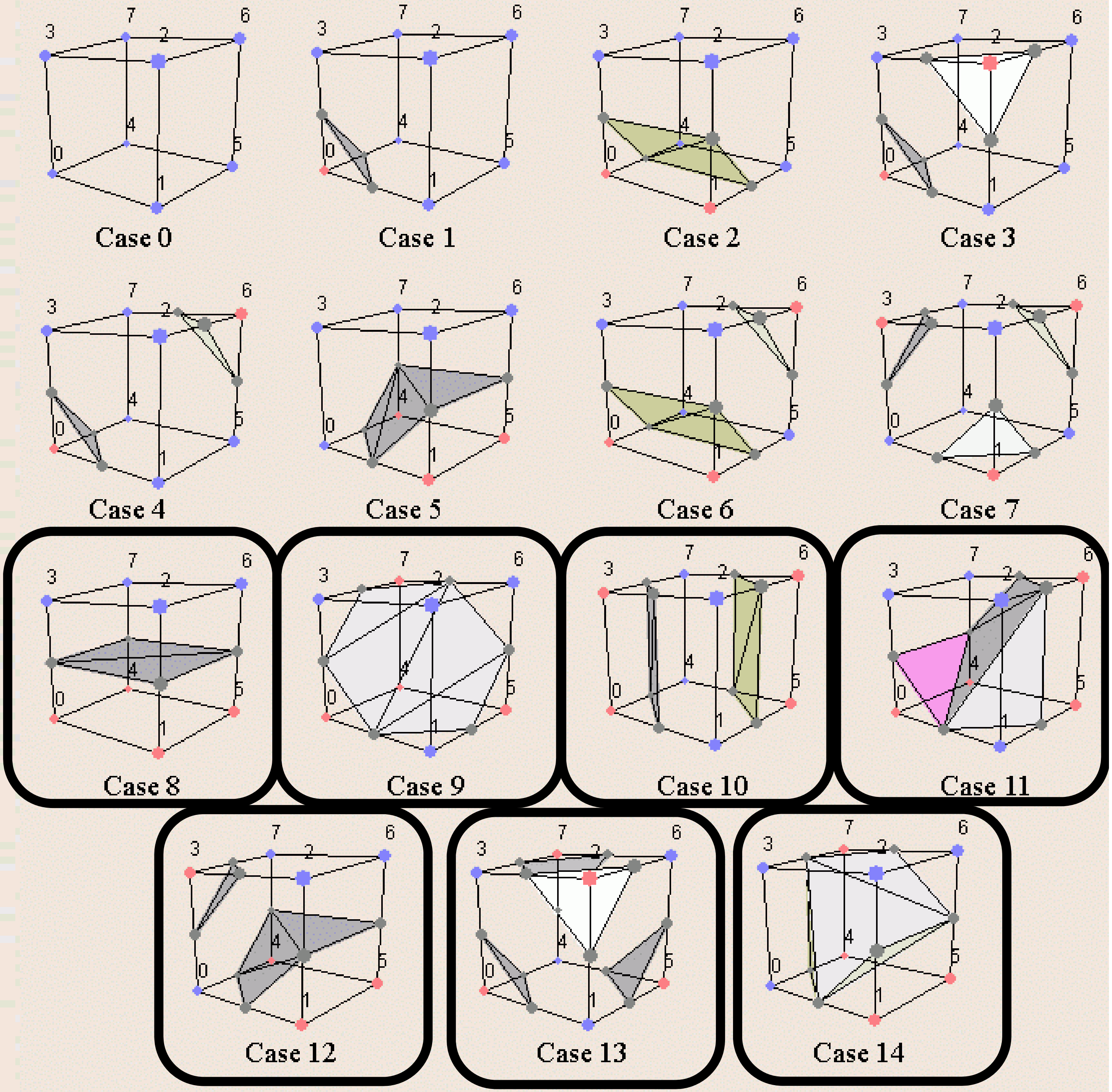
- 5 Above
- 3 Below

3 cases



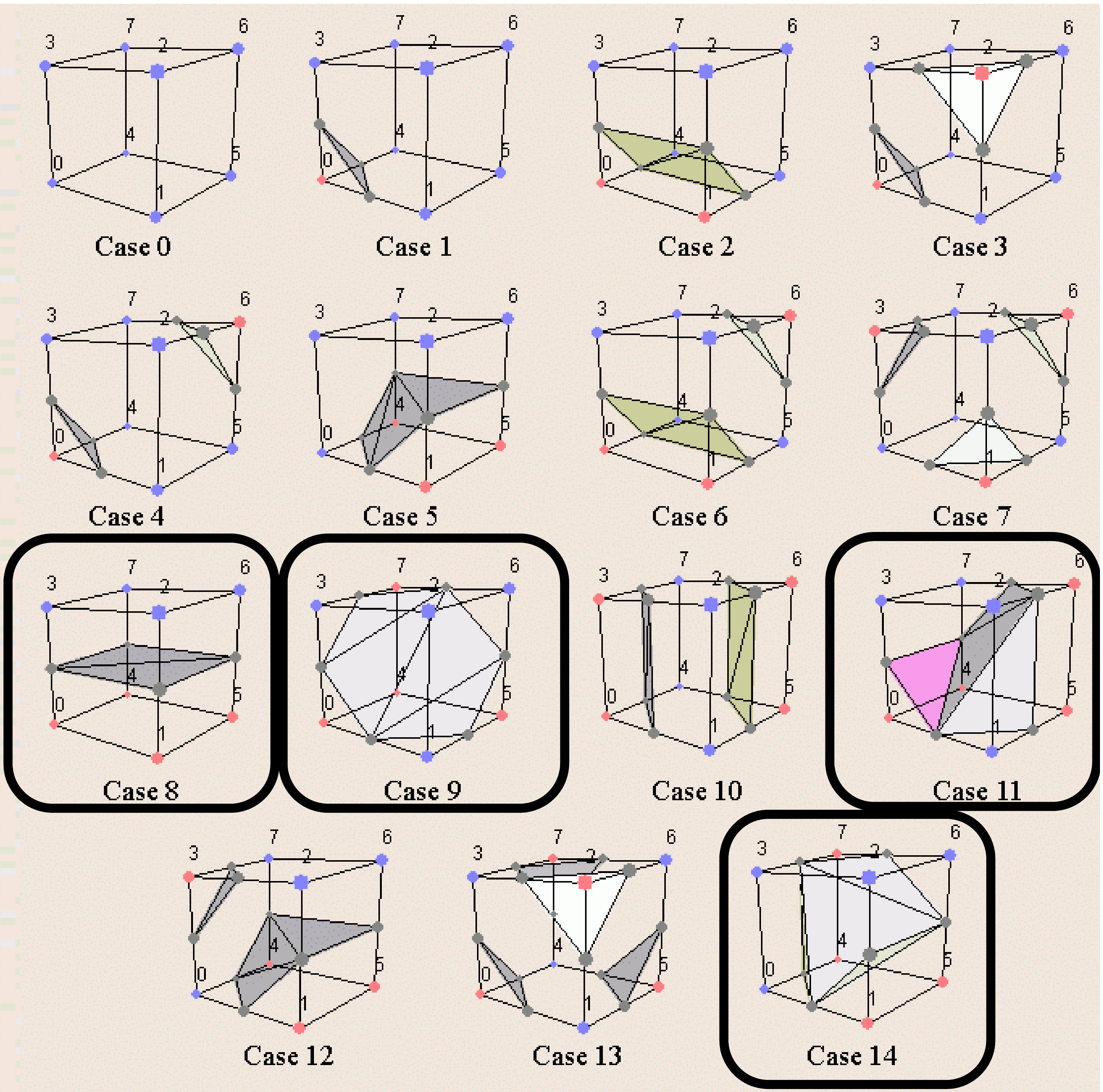
- 4 Above
- 4 Below

7 cases



- 4 Above
- 4 Below

7 cases

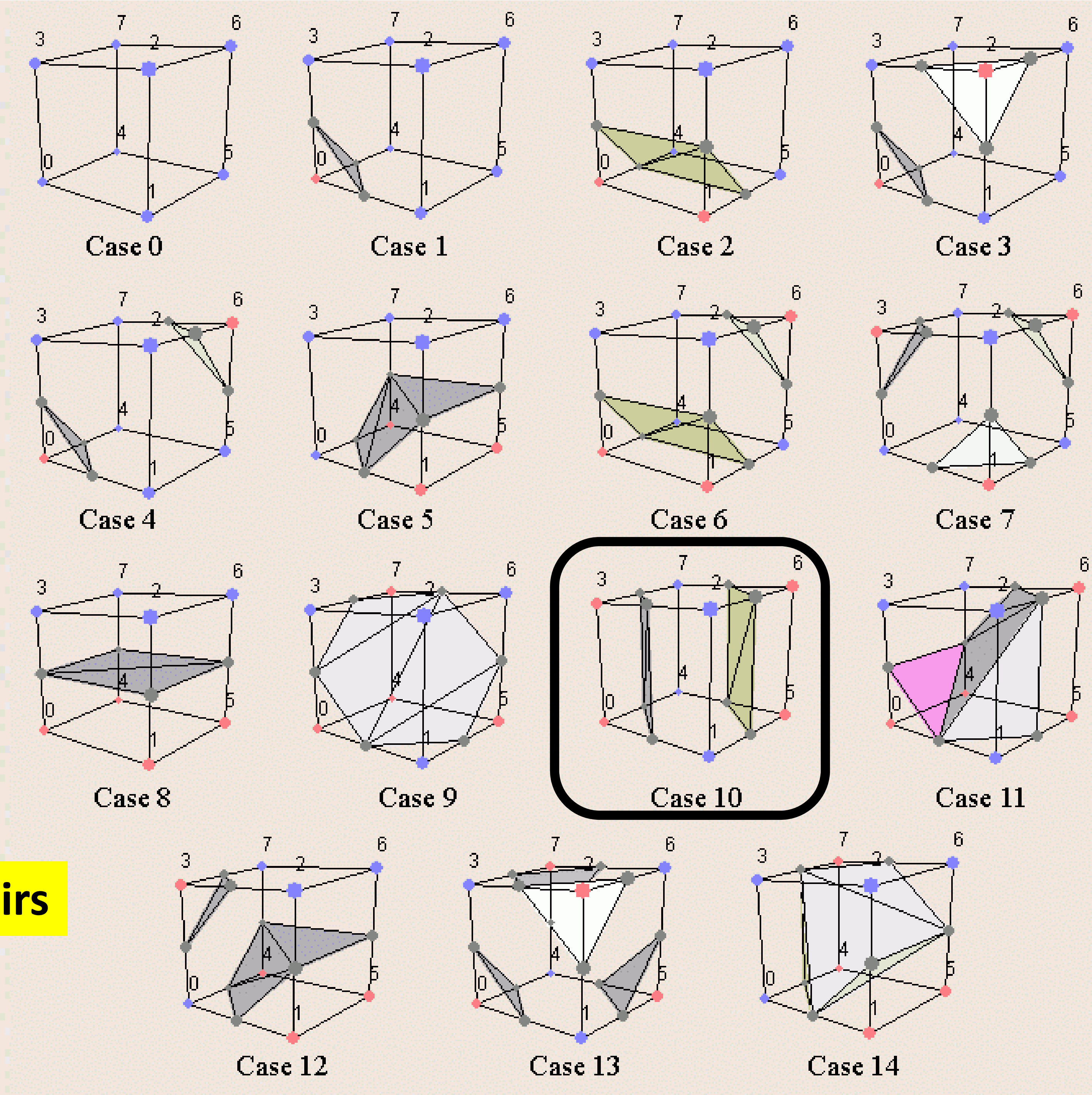


4 connected

- 4 Above
- 4 Below

7 cases

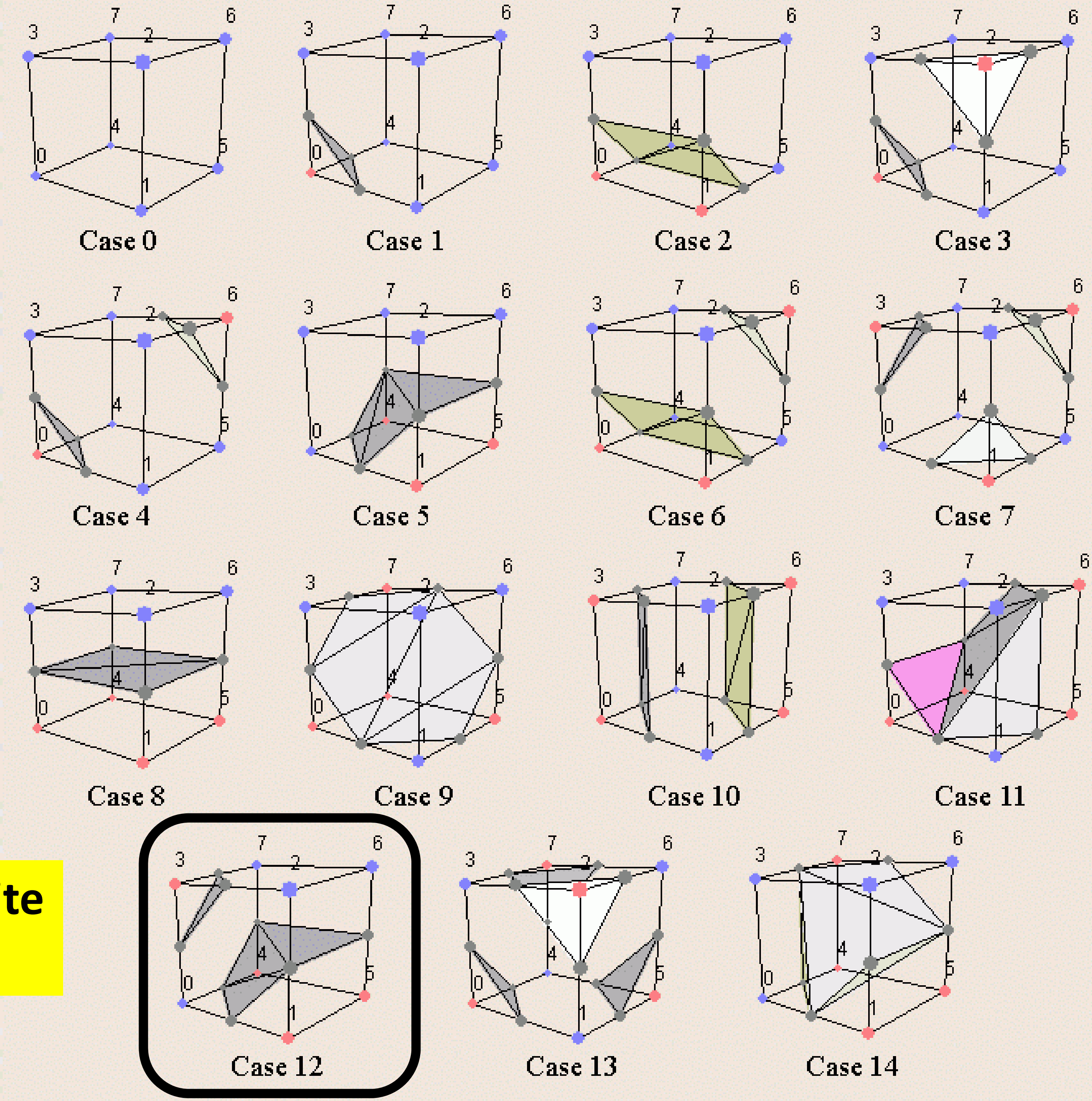
1 opposite pairs



- 4 Above
- 4 Below

7 cases

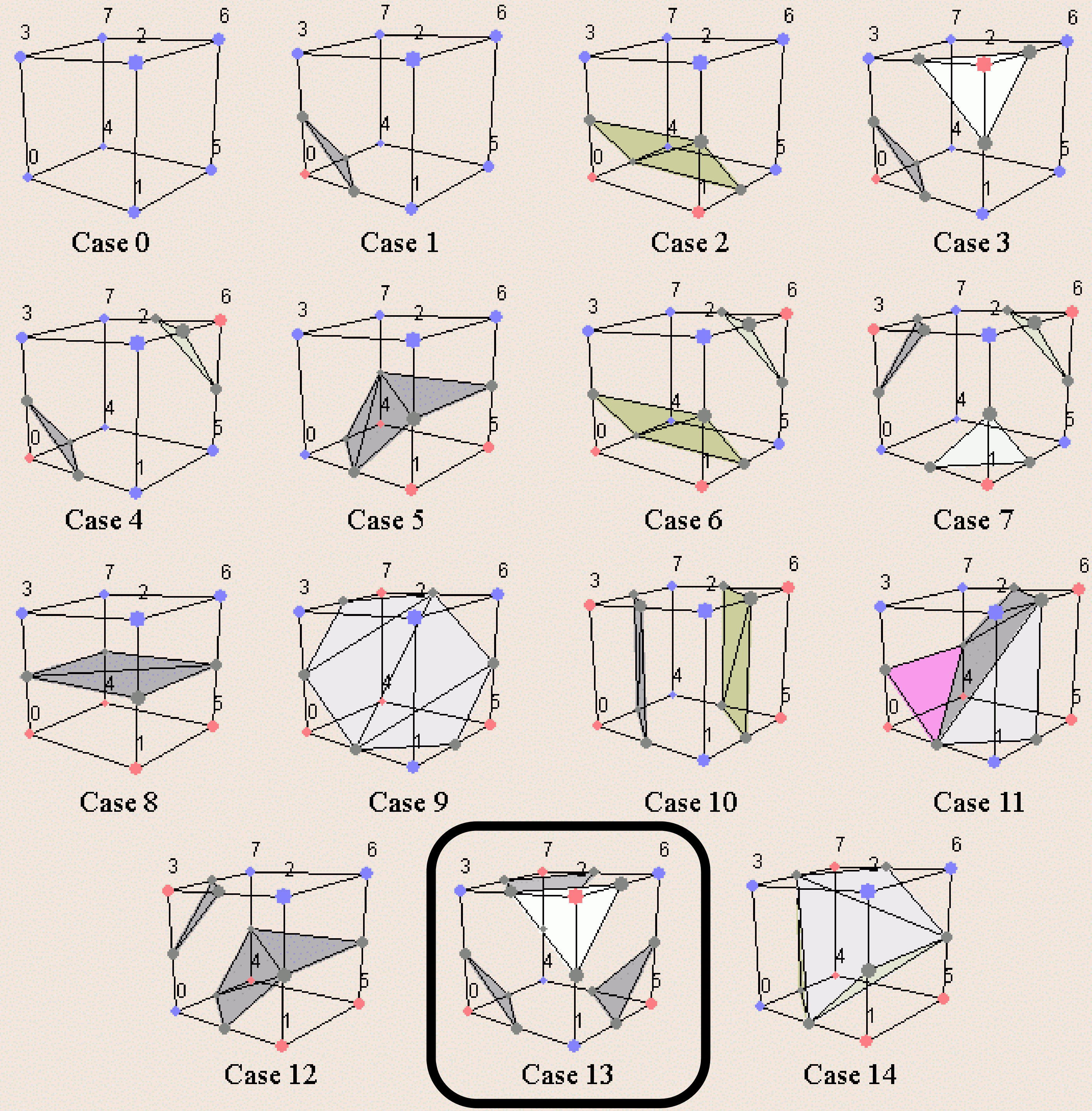
1 vertex opposite to triplet



- 4 Above
- 4 Below

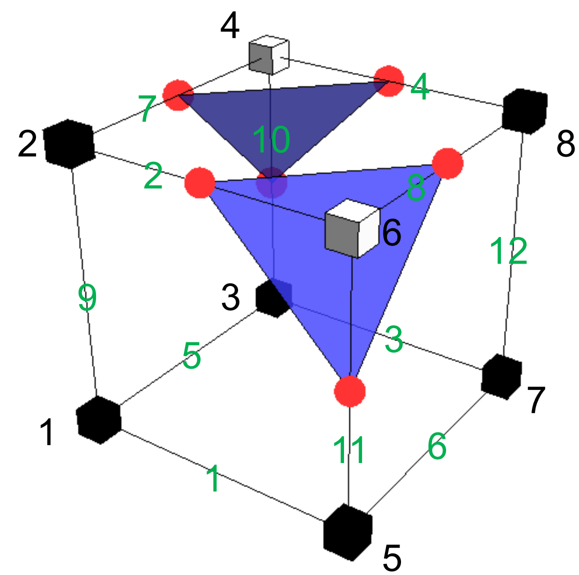
7 cases

1 isolated vertices



Marching Cubes – Look-up Table

- Connecting vertices by triangles
  - **Triangles shouldn't intersect**
  - To be a closed manifold:
    - Each vertex used by a triangle "fan"
    - Each mesh edge used by 2 triangles (if inside grid cell) or 1 triangle (if on a grid face)
    - Each mesh edge on the grid face is shared between adjacent cells
- Look-up table
  - 2^8=256 entries
  - For each sign configuration, it stores indices of the grid edges whose vertices make up the triangles



Sign: "0 0 0 1 0 1 0 0"
Triangles: {{2,8,11},{4,7,10}}

Additional Readings

- Marching Cubes:
  - "***Marching cubes: A high resolution 3D surface construction algorithm***", by Lorensen and Cline (1987)
    - over 17,000 citations on Google Scholar
  - "*A survey of the marching cubes algorithm*", by Newman and Yi (2006)
- Dual Contouring:
  - "*Dual contouring of hermite data*", by Ju et al. (2002)
    - over 800 citations on Google Scholar
  - "*Manifold dual contouring*", by Schaefer et al. (2007)

In VTK

use the **`vtkMarchingCubes`**`()` filter and its function `SetValue(0, iso-value)`