# PV251 Visualization

## Autumn 2024

### Study material

---

## Lecture 5: Spatial data visualization

Spatial data visualization, which includes the entire field of scientific visualization, assumes that the input datasets implicitly contain spatial or spatiotemporal attributes. This fact significantly helps to create and interpret visualizations of such data, because the mapping of data attributes to the graphic ones is intuitive and mostly straightforward. Our visual system constantly receives and interprets images of the physical phenomena that surround us. Therefore, it is relatively natural for humans to process images on the screen in a similar way.

The main differences between the perception of the surrounding world and the spatial data on the screen are:

- When observing the real world, we are not limited to two-dimensional, discrete, low-resolution projection.
- Using the screen, we can visually examine various real and simulated phenomena at different scales.
- On the screen we can dynamically change the contrast, lighting, resolution, density, and other parameters of the displayed data.
- On the screen, we can interactively explore places that are very difficult to access in the real world.
- We can interactively add and delete pieces of data on the screen to improve data context information or remove meaningless data.

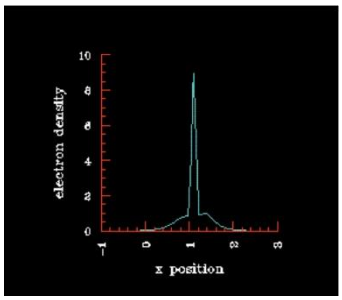
### Mapping the attributes

When visualizing spatial data, we must first decide which spatial data attributes to map to the spatial attributes (locations) on the screen. This phase includes various types of transformations, such as scale, rotation, translation, skew, and projection. Then it is necessary to map the remaining attributes of the input data to other visualization components. These attributes include, for example, color or texture, but also the size and shape of graphic entities.

## 1D data

One-dimensional data is often obtained by sampling a physical phenomenon as it moves along a curve in space.

Let us have a one-dimensional sequence of data with one variable. Then we can map the spatial data to one of the dimensions (axes) of the screen and plot the data values themselves either to the second dimension of the screen (creating a bar graph) or to the color of the marker or region along the first axis (color bar). The data must be scaled so that we can display it over a range of screen dimensions (number of pixels) or within a range of screen attributes (such as the number of colors supported). Parts of the display can be reserved for supporting visualization elements, such as the display of axes, labels, etc.




## Algorithm for drawing 1D data

Suppose we have calculated values ($data_{min}$, $data_{max}$) for the input dataset, which define the minimum and maximum value in this data, and the variable $data_{count}$ determines the number of points (data) to be displayed (it can be all data or just a certain subset) . Next, assume that the part of the screen on which the data will be displayed is a rectangular area defined by ($x_{min}$, $y_{min}$, $x_{max}$, $y_{max}$). The following code can be used to draw a line graph from the points of a field marked as "data".

```
DRAW-LINE-GRAPH(data, dataCount, xMin, xMax, yMin, yMax)
1. dataMin <- computeMin(data, dataCount)
2. dataMax <- computeMax(data, dataCount)
3. xFrom <- xMin
4. yFrom <- worldToScreenY(data[0], dataMin, datamax, yMin, yMax)
5. for i<- 1 to dataCount
6.       do xTo <- worldToScreenX(i, dataCount, xMin, xMax)
7.             yTo <- worldToScreenY(data[i], dataMin, dataMax,
                   yMin, yMax)
8.             drawLine(xFrom, yFrom, xTo, yTo)
9.             xFrom <- xTo
10.            yFrom <- yTo
worldToScreenX(index, dataCount, xMin, xMax)
         return (xMin + index * (xMax - xMin)/dataCount)
worldToScreenY(value, dataMin, dataMax, yMin, yMax)
```

```
        return (yMin + (value - dataMin) * (yMax -
yMin)/(dataMax /         dataMin))
```

The worldToScreenX function is simpler than the worldToScreenY function because we assume that the array of values is indexed from 0. If we wanted to remove this assumption and generalize the function, it would look like this:

```
worldToScreenX(index) {
   return (xMin + (index - indexMin) * (xMax - xMin) / (indexMax -
   indexMin));
}
```

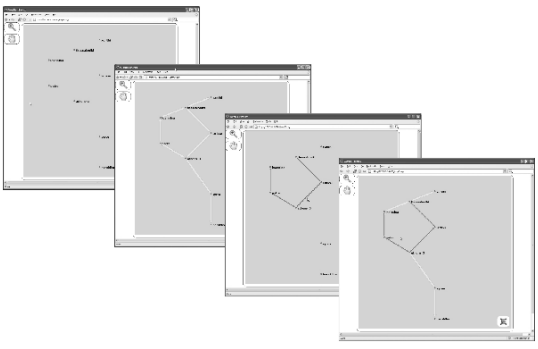
The whole graph drawing function can be improved in various ways. For example, if the number of entry points exceeds the number of available pixels, we can preprocess the entry points (e.g., sample, average…). We may also scale the displayed data to better convey that information.

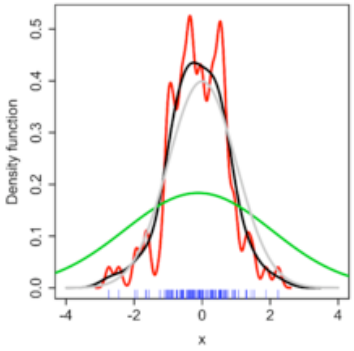
A similar algorithm can be applied to plot data in the form of a color bar, simply replacing line drawing by drawing rectangles (most often of uniform size), the color of which is proportional to the value of the displayed data and the result is transformed into the range of available colors provided by the output device.

## 1D multivariate data

If the input data is marked as multivariate (it contains more variables or more values for one data input), it is possible to extend the previous technique to be able to process such data. This can be achieved by **juxtapositioning** or **superimpositioning**. For line graphs, this means that the visualization will contain a set of isolated graphs (especially in cases where the variables have different scales) or a single graph plotting data on two or more variables. In this case, it's a good idea to draw individual lines in a different style and / or color so that we can distinguish the individual data.

A bar graph for color can be composed similarly. Here is a more intuitive use of the approach, where we layer individual columns on top of each other.





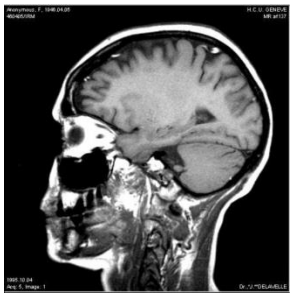
## 2D data

Data containing two spatial dimensions is displayed mainly by mapping the spatial attributes of the data to the spatial attributes of the screen. The result can be one of the following types of visualizations:

1. Image
2. Rubber sheet
3. Cityscape
4. Scatterplot
5. Map
6. Contours

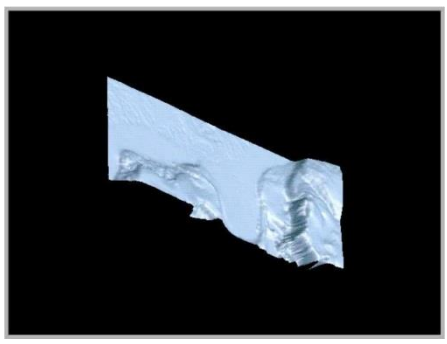
### Image

An image is created as a result of 2D data visualization if one data value in each position is mapped to a color and all intermediate pixels are colored by interpolation. An example is an image obtained from a tomography.



### Rubber sheet

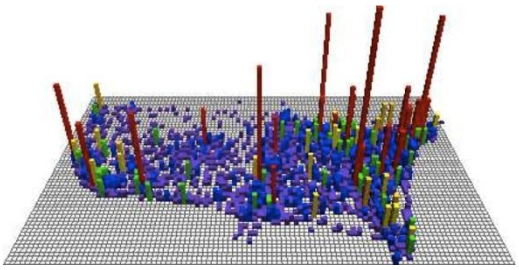
Let us have input data that can be regularly or even irregularly distributed in space. If we map this data to the height of the respective point in three-dimensional space and, in addition, the points are triangulated in such a way that a surface is created, we speak of a visualization of a relief, a surface, or a rubber sheet.



An example is the figure, which shows the sea level and its transition to the mainland in Florida.

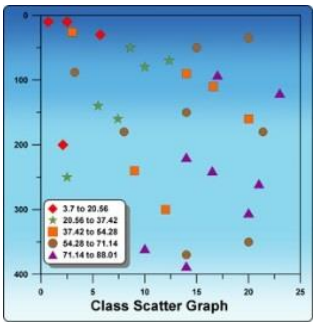
## Cityscape

This type of visualization is created by drawing 3D objects (generally blocks) in various places in the plane. The individual data then controls the attributes of these graphic objects (e.g., height, color). The figure shows an example of such a visualization, where we see the density of air traffic over the United States at a certain point in time.



## Scatterplot

A classic representation, where each data point is placed in the graph, individual data values affect the color, shape, or size of individual marks. It is worth to realize that, unlike the image, there is no interpolation.




## Map

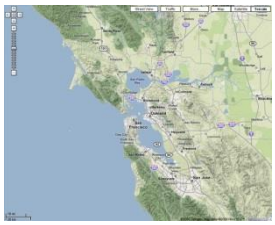
The result of the visualization is a map if the input data contain linear and planar properties, but also point objects.

A linear property can be considered a path or river and is represented as a sequence of connected coordinates, which are drawn as line segments.

Area properties, such as a lake or the area of a state defined by its boundary, are represented by a closed curve - a set of coordinates, where the coordinates of the first and last point are identical. They are usually displayed as a polygon, which can be filled with a color, texture, or repeating symbol.

Point objects (for example, high voltage pylons, schools) are generally represented by a certain symbol placed in a given location. Due to the possibility of overlapping some symbols, the variant of the technique is often chosen, where the symbol does not lie exactly at the corresponding place on the map but is slightly shifted.

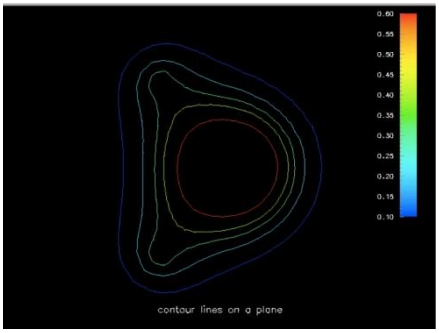
Individual objects usually have their own labels, which again complicates the whole process of displaying the map. We will learn more about the so-called geovisualization in one of the following lectures.



## Contours, isobars

This type of visualization displays boundary information that is derived from an image and represents a certain continuous phenomenon, such as altitude or temperature. It is also referred to as isovalue (e.g., isobar), which means "one value", so the contour in the map determines the boundary between points above and below this value. Each "isovalue" can generate several closed contours. Multiple isobars can be drawn at once, using color, line weight, or markings to distinguish them.

The figure shows the distribution of several contours obtained from a 2D image of a water molecule. Isobar values are distinguished by color.




## 2D multivariate data

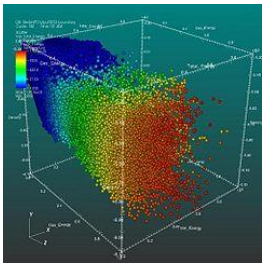
As with 1D data, we can extend these techniques to multivariate data using juxtapositioning and superimpositioning.

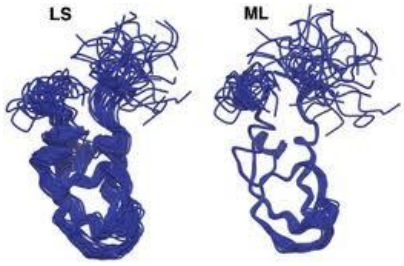
**Juxtapositioning** - simple stacking of several 2D visualizations of one variable into a 3D visualization. This approach allows you to preview all data at once, however, due to overlays, it can be difficult to detect relationships between data. In addition, this technique is limited by the number of variables that can still be clearly displayed, especially for techniques that already use the third dimension (for example, relief or cityscape).

**Superimpositioning** - similarly limited by the number of variables that can be displayed in this way. For example, for reliefs, we draw individual data using translucent surfaces.

Excessive overlap is also not very suitable for maps, as it makes it difficult to extract individual components.

An alternative approach to displaying multivariate 2D data is to use "non-spatial multivariate" visualization, which we will discuss in later lectures.






In addition to visualization techniques that display entire data sets, we can only visualize one-dimensional subsets, projections, or various summaries of data. Then we can use any of the previous techniques. So, let's focus on some of the projection options:

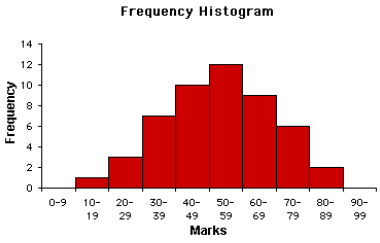
- Frequency histograms

- Merging rows and columns

- Linear "probes"

**Frequency histograms**

Summarization of data sets of an arbitrary dimensionality can be performed by calculating the frequency at which the given values or intervals of values appear in the data. The result is then displayed in the form of a bar chart. If we work with subsets of input values, we must choose the number of individual subsets very carefully and how they are distinguished from each other. If the wrong choice is possible, the loss of important data properties is possible. In this case, the best distribution of the data will be suggested by a person who is familiar with the content of the data. On the other hand, the often-used simple division into a fixed number of equally large subsets is not very effective in this case.




**Merging rows and columns**

A useful mechanism for locating the boundaries of regions of interest and regions with low or high variability is a technique that summarizes the rows and / or columns of an image.

Various techniques are used for this purpose, such as sum, mean, median, standard deviation, maximum or minimum of values. The resulting 1D visualization can then be displayed separately or more often displayed in combination with 2D visualization as additional information.

Color bars, line charts, and bar charts are most commonly used for this technique.

**Linear probes**

A one-dimensional probe for 2D data sets can be compared to a drilled hole in a mineral. A line passes through the input data, and the data that intersects this line is displayed using one of the previous techniques for displaying 1D data. Like a drill bit drilling a hole in a mineral. To achieve this, we use two mathematical tools: parametric equations and bilinear interpolation.

We start by defining the probe using a parametric equation. The definition is based on user input (either a pair of points or a point and direction vector). If we assume that the line is defined by two points P_1 and P_2 (their coordinates), then the parametric equation of the line is defined as

$$P(t) = P_1 + t(P_2 - P_1), \text{ where } 0.0 <= t <= 1.0$$

We can now obtain the coordinates of any point on this line by choosing the value of t. In general, these points on the line are evenly distributed and the number of points is determined in proportion to the length of the line. If we already have the coordinates of the sampled points, we can use interpolation to calculate their values, which can then be visualized as a 1D data set.

## 3D data

As with 2D data, 3D spatial data can be described by vertices, edges, and polygons in the form of discrete continuous patterns or as a structure. In fact, most scientific and technical visualizations involve a combination of these two representations, such as the flow of air around an aircraft wing. We will first focus on exploring the basic techniques for visualizing these types of data, and then discuss the methods that combine these techniques.

The basic techniques for 3D data visualization include:

- Visualization of explicit surfaces
- Visualization of volume data
- Implicit surfaces

## Visualization of explicit surfaces

An explicit surface is a surface that is defined in one of the following ways:

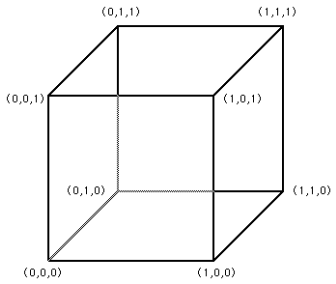
1. List of 3D vertices, list of edges between them (specified as a pair of indexes to the list of vertices), list of planar polygons (usually specified as a fixed or variable list of indexes to the list of edges).

2. A set of parametric equations defining x, y, z coordinates of points on a surface together with a strategy of their connection (for example triangulation), which creates edges and polygons. By determining the step of the parameter in the equations, we can influence the smoothness of the surface.

Example 1:

Let's give an example of a unit cube, which can be represented as follows. Note that each edge is shared by exactly two edges, and each vertex is part of three edges. In addition, each wall has a defined front and rear wall by edge orientation (clockwise or counterclockwise). This condition is especially important for the correct calculation of surface normals.

vertex[0] = (0., 0., 0.)
vertex[1] = (0., 0., 1.)
vertex[2] = (0., 1., 1.)
vertex[3] = (0., 1., 0.)
vertex[4] = (1., 0., 0.)
vertex[5] = (1., 0., 1.)
vertex[6] = (1., 1., 1.)
vertex[7] = (1., 1., 0.)
edge[0] = (0, 1)
edge[1] = (1, 2)
edge[2] = (2, 3)
edge[3] = (3, 0)
edge[4] = (0, 4)
edge[5] = (1, 5)
edge[6] = (2, 6)
edge[7] = (3, 7)
edge[8] = (4, 5)
edge[9] = (5, 6)
edge[10] = (6, 7)
edge[11] = (7, 4)
face[0] = (0, 1, 2, 3)
face[1] = (8, 9, 10, 11)
face[2] = (0, 5, 8, 4)
face[3] = (1, 6, 9, 5)
face[4] = (2, 7, 10, 6)
face[5] = (3, 4, 11, 7)




Example 2:

Another example is the parametric definition of a unit cylinder located in the y-axis.

$y = 1.0, \quad x = \cos\Theta, \quad z = \sin\Theta,$

$0.0 \leq \Theta \leq 2\pi$ (top base)

$y = 0.0, \quad x = \cos\Theta, \quad z = \sin\Theta,$

$0.0 \leq \Theta \leq 2\pi$ (bottom base)

$y = h, \quad x = \cos\Theta, \quad z = \sin\Theta,$

$0.0 \leq \Theta \leq 2\pi, 0.0 \leq h \leq 1.0$ (middle part)

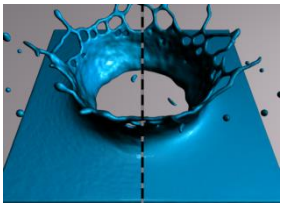
By manipulating the parameter Θ we can change the smoothness of the cylinder surface. If we set the value of h to 1 and the value of Θ to π / 2, then we obtain a cube of height 1.

Other examples of parametric surfaces commonly encountered in computer graphics are Bézier curves and B-splines.

In general, the visualization of spatial data using explicit surfaces depends on whether the input data is associated with vertices, edges, or walls. Examples for each of these cases are:

- Temperature or weight of the node (information about peaks)
- Chemical bond strength (edge information)
- Area coverage by map (wall information)

The displayed information can be mapped to one of the non-spatial graphic attributes - color, transparency, texture, etc.



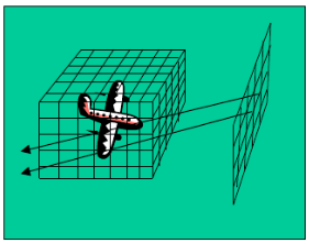
The image comes from the article Explicit Mesh Surfaces of Particle Based Fluids published at the Eurographics 2012 conference.

## Volumetric data visualization

Just as pixels are used for 2D visualization, we use voxels (volume elements) for 3D visualizations. Volumetric data is generally generated by sampling a continuous phenomenon and can be captured by sensors (e.g., tomography) or generated by various simulations (e.g., CFD - computational fluid dynamics). In both cases, we receive multidimensional data with regularly or irregularly distributed positions, and the goal is to provide the user with information about the structure, repeating patterns and anomalies in this data.

Most approaches to volume data visualization fall into one of the following categories:

- **Slicing** - the use of a cutting plane, which can be axially aligned or can have any orientation. The result is a 2D sheet cut from data, which can then be displayed using one of the 1D or 2D visualization techniques.
- **Isosurfaces** - the user specifies the input parameters according to which the description of the data set surface is generated and is visualized using one of the explicit surface visualization techniques.
- **Direct volume rendering** - two methods: the first is based on throwing a beam into a scene containing a solid and calculating the values in individual pixels based on the voxels that the beam intersected. The second method projects each voxel onto the projection plane using one of the methods of accumulating voxel effects on pixels.



**Resampling**

For all the above-mentioned volume data visualization techniques, initial resampling is an essential element.

For example, for isosurfaces we must find places where the data correspond to the selected "isovalue", which in most cases is somewhere between the points of the original data set - a process very similar to creating contours in 2D.

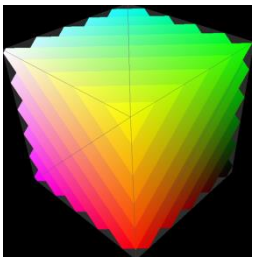
When plating, especially if the clipping plane is not axially aligned with the coordinate system, it is necessary to resample the input data to obtain an evenly distributed set of pixels. In addition, if the input data is unevenly distributed, interpolation must also be used.

In the technique of direct volume rendering, we have to sample data along the rays, which again implies resampling. Resampling is not required only when using parallel projection along the major axes.

Thus, it is clear that the resampling process plays a very important role in the vast majority of volume data visualization techniques.

**Plating of volumetric data by cutting planes**

As with 2D data visualization, we can use the probe technique to create a subset of lower-dimensional input data. A popular technique for volumetric data based on this principle is to use clipping planes, where the data block is "cut" by the plane of the position and the orientation and data that the plane intersects are mapped to the screen.



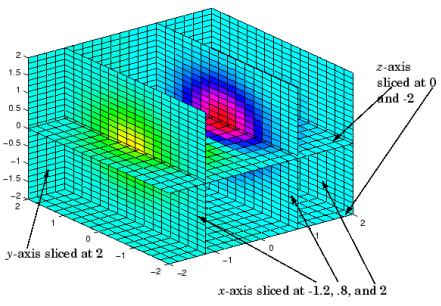
The simplest implementation of this technique limits the orientation of the cutting plane in such a way that the normal of the plane coincides with one of the major axes. The user then specifies the row, column or depth in the data block and the corresponding plate is then displayed using one of the 2D spatial data visualization techniques we have become familiar with earlier.

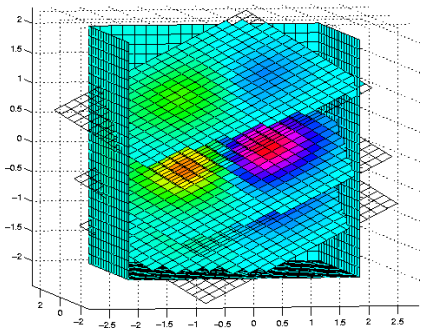
If we remove the restriction on the orientation of the crop plane, then any voxel that is intersected by the crop plane can affect the value in one or more pixels. In this case, we can decide to resample the input dataset at the points where it intersects the clipping plane by placing a regular grid on the clipping plane. Alternatively, we can select the voxel closest to the pixel of the clipping plane, which then represents the volume at this point in the plane. Another approach is to combine the values in the closest voxels to a given pixel on the clipping plane, their contribution being calculated as the weight assigned based on the distance of the center of the voxel from the clipping plane.

To specify the cutting plane, we must set six parameters - three positional and three to define the plane normal.

There are many variations of this technique, each focusing on specific details within the volumetric data. An example might be:

- Non-planar plates
- Sequence of plates with different orientations
- Plates stacked "on top of each other", displayed in parallel
- Plates placed orthogonally, displayed in parallel

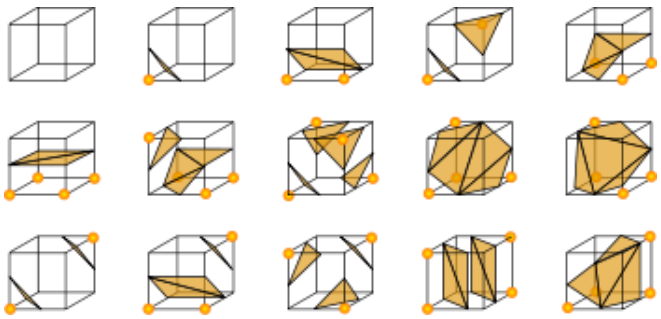


**Obtaining an Isosurface Using Marching Cubes**

The Marching Cubes algorithm was created in 1987 by Lorensen and Cline and focuses on rendering the surfaces of volumetric data. A similar technique was developed one year earlier (Wyvill et al.), however, marching dice have become the most popular algorithm for this problem.

The basis is the definition of a voxel as a cube with values contained in its eight corners. If one or more corners of the cube contain values lower than the user-specified "iso value" and one or more corners also contain values higher than the iso value, it is clear that such a voxel will in some way contribute to the iso-surface. By determining which edges of the cube are intersected by the isosurface, we create triangular patterns that separate the region lying "inside" the isosurface from the region lying outside. The resulting combination of samples from all cubes at the boundary of the iso-surface gives a final surface representation.



We will now describe this algorithm in more detail. The algorithm consists of two main components. The first is responsible for defining the section or sections of the surface that "chop" the individual cubes. If we classify each corner of the cube as falling below or above the defined isovalue, we obtain 256 possible configurations. Two of them are trivial - all the corners of the cube fall in or out. In this case, the cube does not contribute to the iso-surface. For all other configurations, we must determine which edges of the cube intersect the isosurface, and these intersections are then used to create one or more triangular patterns forming the isosurface.

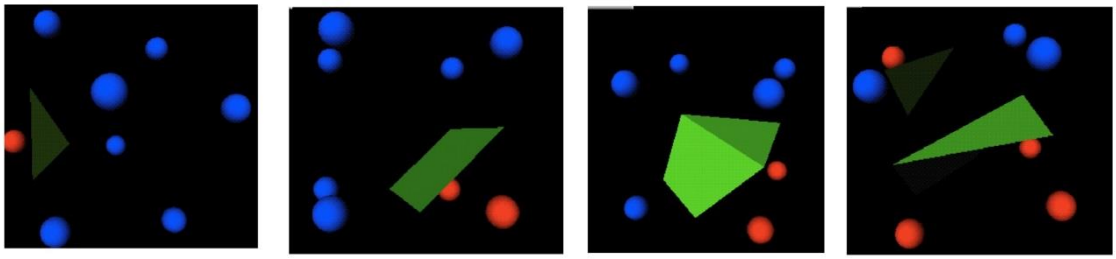
If we consider symmetry, then there are only 14 unique configurations in the remaining 254 formulas. If only one corner contains a value less than the isovalue, a single triangle is created whose vertices lie on the edges adjacent to that corner. The normal of the triangle points from the corner. Thanks to the symmetries, this case generates a total of 8 configurations. One such configuration is shown in the figure on the left.

If the two corners have a value less than the iso value, 3 unique configurations are generated (see figure second from the left). These depend on whether the corners belong to the same edge or to the same wall or are "hit" diagonally.

For three corners with a value lower than the iso value, we again have three unique configurations (picture second from the right). These depend on whether they have 0, 1, or 2 shared edges.

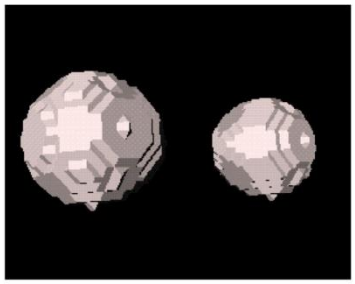
If we have "hit" four corners, we have 7 unique configurations (picture on the right). The result depends on whether the corners share 0, 2, 3 or 4 edges.

Each of the non-trivial configurations adds 1 to 4 new triangles to the iso-surface. The vertices of the triangles can be calculated by interpolation along the edges, or the center of the edge is taken for simplicity. Obviously, the interpolated peaks give us a smoother surface.



The following figure shows a hydrogen atom rendered using a simple version of Marching Cubes, in which the centers of the corresponding edges of the cubes are taken as the vertices of the triangles. The result is also unsatisfactory due to the small input data set.

Now that we can construct a part of the isosurface for one voxel, we can apply this procedure to the entire volume. We can either process each cube separately or we can proceed by taking the cubes that share the edges one by one.



**Problems of Marching Cubes**

One of the obvious problems is the memory requirements when saving the resulting surface. Each boundary cube can generate up to 4 faces. The size of the stored data can be reduced by sharing vertices and edges or even merging coplanar samples into larger areas. Another solution may be to "fit" parametric surfaces to groups of boundary points. However, this approach is difficult for complex surfaces.

Another problem occurs if we do not have a volume completely filled with voxels. This can happen if the data is poorly scanned. The result is holes in the data to which some values must be assigned, most often by interpolation. It is obvious that the newly added values may reduce the validity of the resulting surface.

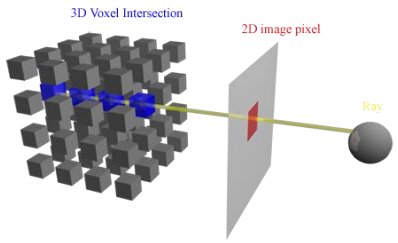
## Direct volume rendering

Direct volume rendering techniques do not use any 3D polygons for rendering. Instead, the pixels of the resulting image are calculated individually, either by throwing a beam into the scene for each pixel or by projecting the voxels onto the projection plane.

The basic process of rendering volumetric data begins with the transformation of the positions of individual voxels into a visual coordinate system, using a standard 3D graphics pipeline. The user must specify the view reference point and view direction vector, the width and height of the image to be projected on the projection plane, and for perspective projection, the distance of the camera from the projection plane must also be specified.

Then we have the option to choose one of the following methods:

- **Forward mapping** - projecting each voxel onto the projection plane and determining which pixels will be affected and how.
- **Reverse mapping** - also called "ray casting". It is the transmission of a ray from each pixel of the projection plane through the volume data, where the values of the data that the ray intersects are sampled and thus determines the resulting value for each pixel.




**Forward mapping - problems:**

F1. How to handle pixels that are affected by multiple voxels

F2. How to handle pixels that have no voxel mapped to

F3. How to deal with the fact that voxels are usually projected into positions between pixels.

**Inverse mapping - problems:**

I1. How to choose the number of points to be sampled along the ray.

I2. How to calculate the value at these points, which often fall between the voxels.

I3. How to combine the points that the ray crossed in its path.

**Solution:**

Problems F2 and F3 can be solved by mapping each voxel to a region of the projection plane, allowing it to partially affect the value of several pixels adjacent to the position at which the voxel was projected. The method using this principle is based on setting the weights of individual voxels for a given pixel, the weight being derived from the distance between the pixel and the projected location of the given voxel. In general, a maximum of four pixels are affected in this way.

Another method using this principle is the so-called "splatting", where we associate a small region of texture with each voxel and this region is projected onto the projection plane.

Problem I1 can be easily solved by determining the spacing between the voxels and setting the sampling frequency to a value less than these spacings. Thus, it is not possible to omit some properties and characteristics of the displayed volume. The sampling itself may be accompanied by resampling or interpolation discussed earlier.

Problems F1 and I3 are commonly solved using a technique known as "compositing". The simplest forms of this technique, such as calculating a maximum volume value or averaging all volume values associated with a pixel / beam. More common approaches assume that each voxel has an associated transparency value and use this when integrating all voxels mapped to a given pixel. If we assume that voxel *i* has color $c_i$ and transparency $o_i$, then its contribution to the resulting pixel value is

$$c_i * o_i * \prod_{j=0}^{i-1} (1 - o_j)$$

In other words, we must determine the accumulated transparency between the projection plane and the voxel and use it to assign the intensity ($c_i$ * $o_i$) of the voxel. The resulting pixel value is then given by the calculation

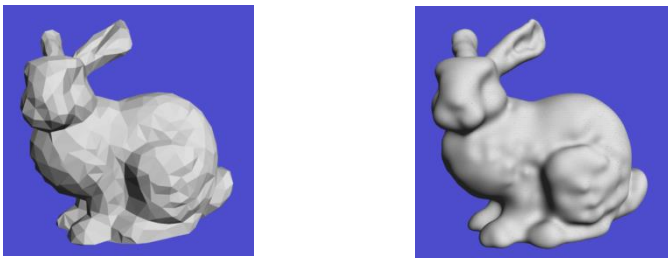
$$I(x, y) = \sum_{i=0}^{n} c_i * o_i * \prod_{j=0}^{i-1} (1 - o_j)$$

## Implicit surfaces

A typical method of modeling surfaces in computer graphics is the use of parametric equations to define points on the surface. These points are then connected into polygonal

networks (mesh). This representation is particularly advantageous for the application of transformations and the calculation of surface normals. An alternative method is to use the so-called **implicit representation**, where the surface is defined as a so-called zero contour (initial contour) function with two or three variables.

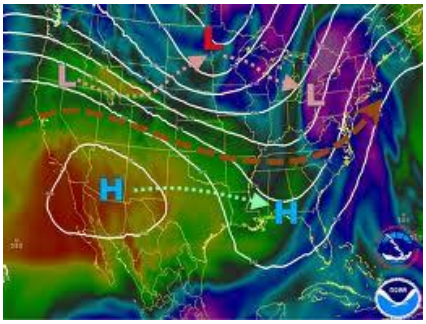
Implicit representations have their strengths when applying operations such as blending or metamorphosis.



In the figure, we see the difference between the polygonal representation of the model and the same model displayed using an implicit surface, which is defined by 800 vertices and their normals.

## Combined techniques

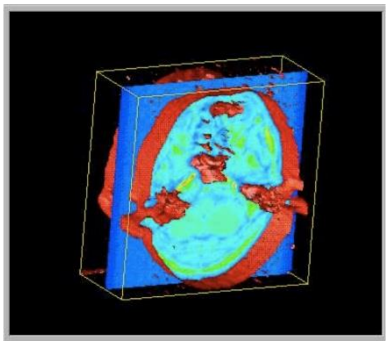
Most effective visualizations are in fact a combination of two or more of the techniques described above. Each of the techniques has its strengths and weaknesses in terms of information that it is or is not able to display effectively. The strengths of the individual techniques can be highlighted by their combination, while of course we must ensure that the overlap is minimized. At the same time, there is increasing pressure to display data simultaneously in several different ways to better understand the information being communicated. For example, the weather forecast includes the display of temperature, wind speed, relative humidity, and several other factors that affect the accuracy of the forecast. In this section, we will focus on visualizations that were created by combining previous techniques.




**Plates combined with isosurface**

The figure illustrates the technique where the surface of medical data is combined with orthogonal slicing of the same data set. The surface is mapped to one color (red) and the resulting values after plating are displayed in blue. The surface can reveal a surface structure

that would be difficult to identify by plating the volume itself. The plate provides detailed 2D information, especially if the color assignment is well chosen. It can provide the user with information about relatively uniform regions as well as those in which dynamic changes are taking place. Another advantage is that the plate also displays the inner (nested) regions in the appropriate range of values, while the general isosurface shows only the outer surface.



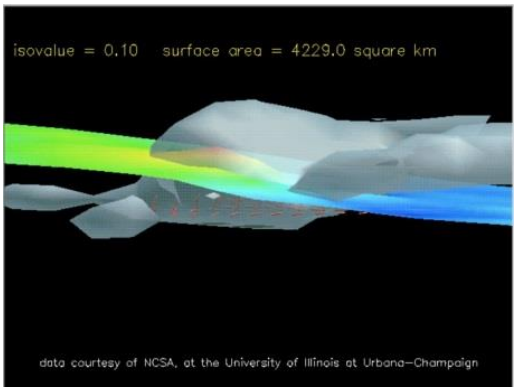
When designing this type of visualization, it is important to consider the following:

- It is not appropriate to support rapid changes in isosurface values
- The position and orientation (along the three axes) of the plate should be easy to control by the user, including the possibility of animating the position of the plate
- To mediate the view of data from all sides, it is necessary to allow the user to change the position and orientation of the camera
- Color assignment is essential - some areas of interest inside the plate can only be discovered by carefully choosing a color scheme. At the same time, it is important to use a different color to display the iso-surface that does not appear in the color map of the plate to avoid misinterpretation.
- The user should be able to easily hide either of these two visualization components, or at least change the transparency.

**Combination of iso-surface and pictograms**

As already mentioned, iso-surfaces are suitable for conveying details of 3D surfaces, but generally do not contain any other aspects of the input data. Pictograms (glyphs), such as a regular arrow, are used to display the magnitude or direction of change in a dataset — either as a gradient in static data or as a flow in dynamic data. Glyphs can be placed near the iso-surface.

The image shows a storm cloud, where the surface shows the density of the water inside the cloud and the arrows show the direction and strength of the wind. In addition, a cutting plane is used to display water density details. By moving the starting positions of the arrows (they are connected to the moving plane) we can reveal "quiet" regions and, conversely, regions with turbulence.




When designing this type of combined visualization, it is advisable to follow similar rules as in the previous case. Interactive control of visualization parameters - isosurface values, basic glyph position, angle of view - contribute to streamlining the visualization. Other aspects are needed to improve the information content of glyphs, such as:

- Changing glyph density
- Glyph size scaling
- Assign color to glyphs

Calculation of the basic position of glyphs based on areas of interest in a vector field
Another potential improvement of this technique may be the use of a cutting plane or a rubber sheet.

## Summary

We got acquainted with several commonly used techniques for visualization of spatiotemporal data. Thanks to the fact that we went through the techniques according to the dimensionality of the data, which we gradually increased, we learned the basics on which the individual techniques are based. Since several different approaches can be used to visualize a particular dataset, it is important to understand the pros and cons of each technique - especially which data properties clearly present the technique. It is often advantageous to combine individual techniques.