Chapter 18

Scientific Visualization Algorithms
Introduction

- Choice of visualization algorithm to be applied depends on:
  - Type of data
  - Desired visual effect

- Example:
  - Given a large scalar data set which must be displayed in its entirety → ray-casting or splatting algorithms
  - To examine areas of equal value more closely → marching cubes algorithm

- Visualization and graphics:
**Introduction (2)**

- Visualization is one level above graphics:
  - Visualization algorithm creates a visualization object from the raw data & specifies its display parameters
  - Graphics algorithms implement these specifications & produce images

- **Visualization object**: a function $V(S)$
  - Domain $S$: space in which the experiment or simulation took place
  - E.g. 1: set of structured points in a 1-, 2-, 3-, or higher-D space; usually referred to as *grid* (most common domain type)
  - E.g. 2: regions of a continuous space
  - E.g. 3: enumerated set
  - Often, the domain will contain a time variable
Introduction (3)

- Range $V(S)$: data items produced by experiment or simulation for elements of the domain
  - *Type* of range items of $V(S)$ distinguishes between visualization methods
  - Common range types: *scalar*, *vector*, *tensor*

- $O$ notation
  
  \[
  O: \text{domtype}_1 \times \text{domtype}_2 \times \ldots \times \text{domtype}_N \rightarrow \text{rangetype}
  \]

- Example
  - Visualization object that represents 2-element vector values (range) on a 3-D grid plus time (domain) has type:
    \[
    X \times Y \times Z \times T \rightarrow \text{vector2}
    \]
    Abbreviation: $O^\text{range\_type}_{\text{domain\_type}}$
  
  So, a 3-element vector field over a 3-D grid is $O^\text{vector3}_{X \times Y \times Z}$
Consider the domain of 3D discrete space $X \times Y \times Z$ as a grid:

- **Regular** $\rightarrow$ elementary volume elements are cubes of the same size
- **Rectilinear** $\rightarrow$ elements are orthogonal parallelepipeds
- **Structured** $\rightarrow$ elements are general parallelepipeds
Introduction (5)

- Regular, rectilinear, and structured grids:

- Alternative: tetrahedral volume elements:
Range values can be mapped onto the grid domain in 2 ways:

- Associated with entire volume elements (*voxels*)
- Associated with grid vertices (*cells*)

To determine the value at an arbitrary 3D point, we have 2 options corresponding to the above mappings:

- The point takes the constant value of the voxel that it belongs to
- Interpolate from the vertex values of the appropriate cell
Scalar Data Visualization

- Two main approaches to visualizing scalar data represented on a grid:
  - S1. To observe one or more surfaces of constant value (isosurfaces) within the field → employ isosurface extraction algorithms
Scalar Data Visualization (2)

- Isosurfaces create sharp renderings & by transforming to a standard representation, they take advantage of widely available graphics techniques to accelerate rendering.
- However, only part of the information present in the scalar field is visible on the isosurfaces.

- S2. Display the entire field by employing a direct volume-visualization technique:
  - Such techniques are slow & generally result in blurry images.
- The choice depends largely on the specifics of the application.
Isosurface Extraction Algorithms

- Often data contain clusters of values which can be separated by surfaces.
- Isosurface algorithms determine these separating surfaces after the user inputs one or more isosurface value(s).
- Once these isosurfaces are established:
  - Quick and easy to display them with standard graphics techniques, as they consist of polygons.
- Marching Cubes & Splitting Box algorithms.
Marching Cubes Algorithm

• Input: Scalar volume data set $O_{X\times Y\times Z}^{\text{scalar}}$ and isosurface scalar value
• Output: list of polygons representing the isosurface
• Marching Cubes (MC) visits every cube of the volume data set
• For each cube, the field values at its 8 vertices are compared to the user-provided isosurface value
• Vertices are thus labeled as 1 (inside, smaller than isosurface value) or 0 (outside, greater than isosurface value)
• Vertex labels are then systematically concatenated & used as an index to a list of pre-computed surface-cube intersections
Marching Cubes Algorithm (2)

- Vertex labeling:
```
Void MC() {
    For (i = 0; i < maxcubeI; i++)
        For (j = 0; j < maxcubeJ; j++)
            For (k = 0; k < maxcubeK; k++) {
                // process cube (i, j, k)
                // label vertices as inside (1) or outside (0)
                l1 = get_label (i, j, k);
                l2 = get_label (i+1, j, k);
                ...
                l8 = get_label (i+1, j+1, k+1);
                // concatenate the 8 labels (++ stands for the
                // string concatenation operator)
                index = l1++l2++l3++l4++l5++l6++l7++l8;
                // map index to one of the 15 basic cases
                // (symmetries) and get required transform
                bindex = map_2_basic_index(index);
                transform = map_2_basic_trans(index);
            }
}
```
Marching Cubes Algorithm (4)

// use bindex to select the appropriate
// precomputed surface-cube intersection
// and reverse transform it
surface_list=
precomputed_surfaces(bindex, transform^-1);
// use interpolation to place the
// intersection surface precisely
for (p=0; p<num_vertices(surface_list); p++)
    compute_precise_edge_position(p, cube_field_values(i,j,k));
// calculate normals at intersection
// surface vertices for rendering
for (p=0; p<num_vertices(surface_list) p++)
    compute_normal(p, cube_field_values(i,j,k));
Marching Cubes Algorithm (5)

- $2^8$ ways to label vertices of a cube:
  - Requires 256 pre-computed surface-cube intersection patterns
  - Reduced to just 15 by taking advantage of:
    - Mirror symmetry
    - Rotational symmetry
    - Inside/outside symmetry
Marching Cubes Algorithm (6)

- Each of the 15 intersection patterns provides the topology of the polygonal intersection surface \textit{with respect to the cube edges}.
- Symmetries used to go from the actual intersection pattern to one of the 15 basic cases form the transform for a cube.
- The exact points of intersection along each cube edge are determined by interpolation:
  - If the edge vertices have associated field values \( v \) & \( v' \) & the isosurface value is \( I (v < I < v') \) \( \rightarrow \) intersection point \( p \) can be expressed as:
    \[
    p = \frac{I - v}{v' - v}
    \]
Marching Cubes Algorithm (7)

- For realistic rendering, normal vectors of the isosurface on the vertices of the resulting isosurface polygons are computed in 2 steps:
  - Compute the gradient vectors of the scalar field at cube vertices
  - Interpolate gradient vectors along cube edges & onto the vertices of the polygons

\[
g_x(i, j, k) = \frac{v(i+1, j, k) - v(i-1, j, k)}{\Delta x},
\]

\[
g_y(i, j, k) = \frac{v(i, j+1, k) - v(i, j-1, k)}{\Delta y},
\]

\[
g_z(i, j, k) = \frac{v(i, j, k+1) - v(i, j, k-1)}{\Delta z}
\]

where \(v(i,j,k)\) & \(g(i,j,k)\) are the field value & gradient vector at cube vertex \((i,j,k)\) and \(\Delta x, \Delta y, \Delta z\) are the differences in the \(x\)-, \(y\)-, & \(z\)-coordinates of the cube vertices involved.
• **MC can be improved in a number of ways:**
  - E.g. avoid re-computation for common edges of neighboring cubes
• **Major disadvantages of MC algorithm:**
  - Large # of polygons created for the isosurface
  - This # is not proportional to the isosurface complexity:
    - Depends primarily on the density of the grid
• **MC can be fully accelerated by the GPU**
Splitting Box Algorithm

- Splitting box (SB) algorithm also creates an isosurface from volumetric scalar data sets:
  - Creates smaller # of polygons than MC by recursively subdividing the original volume **only until** the resulting elements possess a certain complexity property

- Definitions:
  - *Box*: Rectangular parallelepiped with edges parallel to the main axes of the grid
  - *Length* of an edge: # of grid vertices it contains
  - An edge has the *MC property* if it contains at most one isosurface transition
    - SB algorithm uses this generalized property to end the subdivision process as early as possible in the box hierarchy
A box Face is MC if its 4 edges are MC
A Box is MC if its 6 faces (or twelve edges) are all MC

```c
void SB (box); {
    if  MC_property(box)
        generate polygons using the 15 cases of MC;
    else if  size(box)=2^3
        generate polygons by analytical processing;
    else {
        subdivide box along longest edge into box1 and box2;
        SB(box1);
        SB(box2);
    }
}
```
Splitting Box Algorithm (3)

- Box subdivision:
Direct Volume Visualization

- 3-D scalar data sets consisting of sampled data or representing amorphous phenomena are hard to represent using surfaces.
- In such cases we can employ visualization algorithms that display the data by directly interrogating the data set.
- Volume rendering:
Direct Volume Visualization (2)

- Two types of algorithms
  - *Backward projection (ray casting):*
    - Fires rays for each image pixel into the data set, obtains samples and combines them into a final color
  - *Forward projection (splatting):*
    - Projects each voxel in the data set onto the image plane and establishes which pixels it affects using a filter
Ray Casting

• Ray casting consists of 3 steps:
  - Classify each voxel according to its content
  - Transform rays or data so that they are aligned with the viewing direction
  - Combine the result along each ray

• **Classification step:**
  - Classifies each voxel depending on its material content
  - Result is a color and transparency value

• **Example:**
  - In a medical scanning application color & transparency values are assigned according to the x-ray absorption values
  - Set of possible materials: material = { air, fat, soft-tissue, bone }
  - Material $i$ has an a-priori given probability distribution $P_i(I)$ to have intensity value $I$ in homogeneous form
  - Which material(s) does a voxel contain & in what proportion?
Ray Casting (2)

- If $P(I)$ denotes the probability that a voxel has intensity value $I$:
  $$P(I) = \sum_{i=1}^{m} \rho_i P_i(I)$$
  where $\rho_i$ is the proportion of material $i$ in the voxel

- Bayesian estimate of amount of material $i$ in voxel with intensity $I$:
  $$\rho_i(I) = \frac{P_i(I)}{\sum_{j=1}^{m} P_j(I)}$$
  $$\sum_{j=1}^{m} \rho_j(I) = 1$$

- Color/transparency $C$ of the voxel computed as: $C = \sum_{i=1}^{m} \rho_i C_i$
  where $C_i = (\alpha_i R_i, \alpha_i G_i, \alpha_i B_i, \alpha_i)$ is the color and transparency value that corresponds to material $i$

- Material intensity probability distributions:

![Material intensity probability distributions diagram](image)
Ray Casting (3)

- **Transformation Step:**
  - Two ways of aligning to the viewing direction:
    - *Ray transformation*
    - *Data transformation*

**Ray Transformation:**
- Casts rays into the volume data & takes samples at equidistant points along each ray
- Samples are computed by tri-linear interpolation from the 8 nearest voxel values
Ray Casting (4)

- **Example:** $v_{i,j,k}$, $i, j, k \in \{-,+,+\}$ represent the values of the 8 surrounding voxels for a sampling point $s$ & $d_x^-, d_y^-, d_z^-$ represent the distances from $s$ to the centers of the 3 voxels with the smaller indices in each axis as a portion of the inter-voxel distance; the interpolated value at $s$ is:

$$v_s = (1-d_x^-)[(1-d_y^-)[(1-d_z^-)v_{---} + d_z^-v_{--+}]+d_y^-[(1-d_z^-)v_{---} + d_z^-v_{+++}]]$$

$$+d_x^-[(1-d_y^-)[(1-d_z^-)v_{---} + d_z^-v_{--+}]+d_y^-[(1-d_z^-)v_{---} + d_z^-v_{+++}]]$$

**Data Transformation:**

- Aligns the volume data with the viewing direction
- If $z$ is the viewing axis $\rightarrow$ voxel data are realigned so that voxels with the same $z$-coordinate lie on the same viewing ray
- Data transformation achieved by a shear-warp operation
- Slices of the volume data are sheared in $xy$-plane by factors $s_x$ & $s_y$
- Data of each slice are re-sampled using bi-linear interpolation
Ray Casting (5)

- Shear matrix for a parallel projection is:

\[
SH_{\text{par}} = SH_{xy}(s_x, s_y) = \begin{bmatrix}
1 & 0 & s_x & 0 \\
0 & 1 & s_y & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

- In the case of a perspective projection also need to scale each slice \(s\) is determined from the viewing transformation:

\[
SH_{\text{pers}} = \begin{bmatrix}
1 & 0 & s_x' & 0 \\
0 & 1 & s_y' & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & s & 1
\end{bmatrix}
\]

- A volume data slice at \(z=z_0\) is thus scaled by \(1/(1+s \cdot z_0)\) to return to normal homogeneous form

- **Problem**: Non-homogeneous sampling of volume data caused by the diverging rays (sampling density is a function of depth)
Ray Casting (6)

- **Solution**: Adaptively introduce extra rays as a function of depth
  - Rays then get sub-sampled in slices \(\rightarrow\) end up with initial ray resolution
- Shear operation for parallel (left) & perspective (right) projection:

![Diagram of viewing rays, shear operation, and shear & scale](image)
Ray Casting (7)

**Combination Step:**

- Determines the resulting color value along each ray by:
  - combining the value at successive sample points (ray transformation) or
  - successive voxels along the Z-direction (data transformation)
- Both cases are referred to as *ray samples*
- Value at each ray sample stored in RGBA form → traverse ray samples from *back-to-front*
- At each ray sample compute outgoing color value $RGB_{out}$ as a function of incoming value $RGB_{in}$, and the color & transparency properties of current ray sample:
  \[
  RGB_{out} = RGB_{in} (1 - \alpha_{current}) + RGB_{current} \alpha_{current}
  \]
  where $\alpha_{current}$ ranges from 0 (totally transparent) to 1 (totally opaque)
Ray Casting (8)

- Combining the ray samples:
To allow for early ray termination, we can reverse the process to \textit{front-to-back}.

- Must accumulate color and transparency values

\[ RGB_{\text{out}} = RGB_{\text{acc}} + (1 - \alpha_{\text{acc}})RGB_{\text{current}} \]
\[ \alpha_{\text{acc}} = \alpha_{\text{acc}} + (1 - \alpha_{\text{acc}})\alpha_{\text{current}} \]

where \( RGB_{\text{out}} \) is the outgoing color in the front-to-back direction.
Splatting

- The splatting algorithm:
  - Gives voxels the priority
  - Considers each voxel’s projection onto the image plane (i.e. the pixels)
  - Appears like the voxel was “thrust onto” the image plane
- The discrete voxel space represents a continuous volume
- Values \( f(x, y, z) \) can be reconstructed by:
  \[
  f(x, y, z) = \sum_i \sum_j \sum_k v(i, j, k) \cdot h(x - i, y - j, z - k),
  \]
  where \( v(i, j, k) \): the discrete voxel values
  \( h \): the reconstruction kernel
- The summation is taken over a 3D volume
  - Volume's size is equal to that of the reconstruction kernel
Splatting (2)

- Consider the contribution \( \text{contri}(x,y,z) \) of a voxel \((i,j,k)\) to a point \((x,y,z)\)

\[
\text{contri}(x, y, z) = v(i, j, k) \cdot h(x - i, y - j, z - k)
\]

- Assume \( z \) to be perpendicular to the image plane:

\[
\text{contri}(x, y) = \int_{-\infty}^{\infty} v(i, j, k) \cdot h(x - i, y - j, z)dz
\]

- \( v(i,j,k) \) does not depend on \( z \) and can be taken outside the integral:

\[
\text{contri}(x, y) = v(i, j, k) \int_{-\infty}^{\infty} h(x - i, y - j, z)dz
\]
Splatting (3)

- A kernel is centered at every voxel.
- Its contributions to image-space pixels can be determined by projecting the kernel onto image space.
- All kernels have the same projection, called footprint:

\[
\text{footprint}(\alpha, \beta) = \int_{-\infty}^{\infty} h(\alpha, \beta, z) \, dz
\]

where \( \alpha, \beta \): represent the image-space \( X \)- and \( Y \)-displacement from the central pixel of the kernel projection.

- If the image plane is not aligned with the axes of the voxel volume, then the footprint function is slightly more complicated.
The voxels are processed in **sheets**

Sheet: a plane of voxels parallel to the image plane

If the image plane is not aligned with the voxel space axes:
- Define the sheets by the pair of voxel axes most parallel to the image plane
Splatting (5)

• Algorithm: Processing starts with the sheet nearest to the observer
  - front-to-back allows for early termination

for each sheet s front-to-back
  for each voxel (x, y, s) in sheet s
    for all footprint offsets (a, b) {
      frame_buffer(x+a, y+b) =
      frame_buffer(x+a, y+b) +
      voxel(x, y, s)*footprint(a, b)*
      (1-transparency_buffer(x+a, y+b))

      transparency_buffer(x+a, y+b) =
      min(1, transparency_buffer(x+a, y+b) +
          transp(voxel(x, y, s))
      *footprint(a, b))
    }

Graphics & Visualization: Principles & Algorithms
Chapter 18
37
Splatting

- Splatting
  - Requires more computation as it processes the entire voxel space
  - Does not take advantage of bounding volumes
  - Voxels are processed independently
  - Is more amenable to parallel implementation than ray-casting
  - Very easy to integrate a multi-resolution representation of the volume data into the rendering stage
Vector Data Visualization

- Vector fields are common results of experiments and simulations
  - electromagnetic fields
  - derivatives of scalar fields
  - wind-velocity data
- Extra complexity as each element has several dimensions
- The dimensionality of the definition grid and that of the vectors in the field are independent

\[
O^{\text{vector}3}_{X \times Y} \quad \text{and} \quad O^{\text{vector}2}_{X \times Y \times Z} : \text{possible fields}
\]
\[
O^{\text{vector}2}_{X \times Y} \quad \text{and} \quad O^{\text{vector}3}_{X \times Y \times Z} : \text{more frequently}
\]
Vector Data Visualization (2)

- **Static** (or steady) vector fields
  - Do not change over time
  - E.g. the constant gravitational field between a set of static objects

- **Dynamic** (or unsteady, time-varying) vector fields
  - Represent a dynamic phenomenon
  - E.g. wind velocity and direction data during a day
  - Represented by “snapshots” of the field at discrete points in time
  - Each snapshot ≡ static vector field
Vector Data Visualization (3)

- Function types of 3D vector fields over 3D grids

\[ V_{\text{static}} : O^{\text{vector3}}_{X \times Y \times Z}, \]
\[ V_{\text{dynamic}} : O^{\text{vector3}}_{X \times Y \times Z \times T} \]

the last parameter in \( V_{\text{dynamic}} \) refers to time
Hedgehogs

- Representation of vectors as arrows
- Extensively used to represent vector fields
- Arrow length & direction $\rightarrow$ corresponding quantities of vector
- Dynamic vector fields can be represented by animations
  - Each frame is the arrow plot of the field at a specific time instant
- Major problems:
  - Visual clutter
    - In the case of dense fields
  - Projective distortion (foreshortening)
    - When higher dimensional fields are projected to 2D for display purposes
Hedgehogs (2)

Example $O_{X \times Y}^{\text{vector2}}$
Hedgehogs (3)

Example $O_{X \times Y \times Z}^{\text{vector}^3}$
Particle Advection

• Imagine the trace of single particles through a vector field
  ■ think of a wind-tunnel experiment
  ■ we can release a small and light ping-pong ball and observe the path that it takes
  ■ better, release a small number of colored balls at different points and observe their joint behavior

• Rely on the existence of flow patterns in vector fields

• Display the effect of these flow patterns on weightless and frictionless particles that are advected through the field from their initial position
Particle Advection (2)

- **Visualization point** is a triplet:
  \[ \text{vispoint} : [X, Y, Z] \]

- **Visualization line** is a set of points (not necessarily a straight line):
  \[ \text{visline} : \{\text{vispoint}\} \]

- **Streamline** is a function
  - takes a static vector field and a set of initial points
  - produces a set of visualization lines, one for each point:
    \[ \text{streamline} : V_{\text{static}} \times \{\text{vispoint}\} \rightarrow \{\text{visline}\} \]
  - is the trace of a particle in a static vector field \( \vec{S} \)

- If \( \pi \) is the parameter of the streamline and
  \( s \) is a point on it:
  \[ \frac{ds}{d\pi} = \vec{S}(s(\pi)) \]
• **Twist** of a vector field along streamlines
  - Visualized by plotting ribbons
  - **Ribbons** are the result of connecting the traces produced by pairs of neighboring particles
• Can plot an extra parameter by color coding its values along the length of the streamline
• Can use streamlines to visualize a single time instant $t$ of a dynamic field $\vec{D}$

$$\frac{ds}{d\pi} = \vec{D}(s(\pi), t)$$
Particle Advection (4)

- Streamlines and Ribbons for Static Vector fields
Particle Advection (5)

- Streaklines
  - To visualize a dynamic vector field
  - Plot paths of particles as they are advected through the field
  - The field applied to particles at each step is a function of time
    Must first define pathline
Particle Advection (6)

- **Pathline**
  - A function: given an initial point \( s_0 \) at an initial time \( t_0 \)
  - produces the trace of the point through a dynamic field
    \[ \text{pathline} : V_{\text{dynamic}} \times \text{initial} \rightarrow \text{visline} \]
    where initial = (vispoint, time)

- Successively replaces the initial point by the result of advecting it through each instance of the field:
  \[
  \frac{dp(\text{init}, t)}{dt} = \bar{D}(p(\text{init}, t), t)
  \]
  where \( p(\text{init}, t) \): the pathline starting at \( \text{init} = (s_0, t_0) \)
  at parametric time \( t \)
Particle Advection (7)

- Streakline can be viewed as a function that takes a dynamic vector field and a set of initial points at given initial times and produces a set of visualization lines, one for each point

\[
\text{streakline} : V_{\text{dynamic}} \times \{\text{initial}\} \rightarrow \{\text{visline}\}
\]

where initial = (vispoint, time)

- Streaklines do not necessarily start at the beginning of the simulation
Line Integral Convolution

- Particle advection techniques present only a very small portion of the information and can lead to wrong conclusions about a field.

- Line integral convolution (LIC)
  - Allows the global visualization of dense static vector fields over 2D or 3D grids.
  - An input texture with resolution equal to the cell count of the grid is “locally” blurred to produce an output texture of the same size.

- The vector field is visualized via its blurring effect on the texture.

- The input texture may be
  - related to the vector field
  - completely unrelated, such as a noise image.
Line Integral Convolution (2)

- Assume an $O_{X\times Y}^\text{\text{vector2}}$ vector field
- The local behavior of the vector field can be approximated by computing a streamline that
  - starts at the center of a cell [pixel] $(x,y)$
  - extends in both the positive and the negative directions of the field
- The value at pixel $(x,y)$ of the output image is computed by
  - A convolution of a 1D strip of the input image with a 1D convolution filter
  - The convolution is performed over $L$ cells of the streamline of $(x,y)$ in each direction
Line Integral Convolution (3)

- If $O$ and $I$ are the output and input images, the LIC function is:
  \[ O(x, y) = \sum_{p \in A} I(p) \cdot h(p) \]

  where $A$: the set of cells of the streamline within a discrete cell distance $L$ from $(x,y)$

- $L$ is half the length of the convolution kernel
  \[ h(p) = \int_{\lambda_1}^{\lambda_2} k(w) dw \]

  where
  - $\lambda_1$: the arclength of the streamline from $(x,y)$ to the point where it enters cell $p$
  - $\lambda_2$: the arclength of the streamline from $(x,y)$ to the point where it exits cell $p$
  - $k(w)$: the convolution filter function
Line Integral Convolution (4)

- The length of the convolution kernel (2L) is a critical parameter
  - large $L \rightarrow$ LIC functions of most cells have similar values (more blurring)
  - small $L \rightarrow$ insufficient filtering (no blurring)
Line Integral Convolution (5)

- So far LIC was used to visualize vector **direction**
- To visualize **magnitude**
  - Vary $L$ according to local vector magnitude
  - Amount of blurring is proportional to vector magnitude
- Generalization of LIC to $O_{X \times Y \times Z}^\text{vector3}$ fields:
  - Number of voxels in such 3D fields can be very large
  - Computational cost can be proportionally high
  - Hard to visualize such a dense 3D output
  - Define a 3D ROI by placing constraints on some scalar value of the field
  - Use these constraints to mask out (set to 0) parts of the 3D input texture
  - LIC calculations performed on voxels that correspond to non-zero input texture elements
Line Integral Convolution (6)

for each voxel v
    if (scalar_test(v) ∉ ROI) set input_texture(v) to 0
for each voxel v
    if (input_texture(v) ≠ 0) compute LIC function output 0(v)
Visualization of Vector Field Topology

• Minimizes the visual clutter from the visualization of vector fields
• Concentrate on important field features (visualize its topology)
• Critical points:
  - The points where the field has zero value
  - The most important topological elements
  - Represent singularities of the vector field
  - Classified according to the eigenvalues of the Jacobian matrix of the field evaluated at their position
• For a 2D vector field \( \vec{S}(x, y) = (S_x(x, y), S_y(x, y)) \) the Jacobian matrix is

\[
J_2 = \begin{bmatrix}
\frac{\partial S_x}{\partial x} & \frac{\partial S_x}{\partial y} \\
\frac{\partial S_y}{\partial x} & \frac{\partial S_y}{\partial y}
\end{bmatrix}
\]

- Has 2 eigenvalues that are either both real or both complex
Visualization of Vector Field Topology (2)

Saddle point
\[ R_1 \cdot R_2 < 0 \]
\[ I_1, I_2 = 0 \]

Attracting focus
\[ R_1, R_2 < 0 \]
\[ I_1, I_2 \neq 0 \]

Attracting node
\[ R_1, R_2 < 0 \]
\[ I_1, I_2 = 0 \]

Center
\[ R_1, R_2 = 0 \]
\[ I_1, I_2 \neq 0 \]

Repelling focus
\[ R_1, R_2 > 0 \]
\[ I_1, I_2 \neq 0 \]

Repelling node
\[ R_1, R_2 > 0 \]
\[ I_1, I_2 = 0 \]
Visualization of Vector Field Topology (3)

- **Separatrices**
  - Boundaries between field sectors with different type of flow
  - Curves (2D fields) and curves or surfaces (3D fields)

- **Skeleton**
  - A graph of the critical points and the separatrices of a vector field
  - Provides a simplified representation of the field
Scalarization

- Simple but lossy
- Discards most components of the vector field creating a scalar field
  - Can be displayed by a simple color-value association
- E.g. pick the magnitude component of the vector field for display and discard all directional information
- Hall’s color mapping method
  - Does not discard vector components
  - Maps direction and magnitude to different color characteristics
Hall’s method:

- Assume vectors of $O_{X \times Y}^{vector2}$ represented in a polar coordinate system as triplets $(\rho, \theta, \phi)$
- Map $\rho, \theta, \phi$ to a spherical color-coordinate system
- $\theta$ can represent hue
- $\phi$ can represent tone (shade)
- $\rho$ can represent purity

Thus:
- pure colors represent vectors of maximum magnitude
- direction is mapped to hue and tone
Scalarization (3)

- Hard for our brain to associate color with vector characteristics
- Improved if the colors are quantized before display
  - helps us to classify the vectors into a few major direction categories
Vector Field Simplification

- Simplification techniques:
  - Can be used to simplify (reduce) the field before visualizing it
  - Reduction of the number of vectors in a controlled way which aims to preserve important field properties
- The risk of missing important data is reduced
- Generalization of edge-collapse operation:
  - For vector fields defined over tetrahedral meshes
  - Reduces the tetrahedra that share a collapsed edge to triangles
  - These are deleted
  - Tetrahedra that share a vertex of the collapsed edge must have this vertex updated
Vector Field Simplification (2)

_Simplification Algorithm:_

- Create a queue of candidate edge-collapses by ordering all edges of the tetrahedral mesh according to an error metric that measures the degradation of the field as a result of each collapse.
- Repeat until the simplification target is reached:
  - Remove the edge from the front of the queue,
  - Apply the associated edge-collapse,
  - Re-compute the error metric for affected queue elements and reorder the queue.
### Vector Field Visualization Techniques: Overview

<table>
<thead>
<tr>
<th></th>
<th>Hedgehogs</th>
<th>Particle advection</th>
<th>LIC</th>
<th>Vector field topology</th>
<th>Scalarization</th>
<th>Simplification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full field visualization</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direction visualization</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x^1</td>
<td>x</td>
</tr>
<tr>
<td>Magnitude visualization</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x^1</td>
<td>x</td>
</tr>
<tr>
<td>No significant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>information loss</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avoids clutter</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

1 Either vector direction or magnitude is usually visualized with this technique.