Direct Volume Rendering

Balázs Csebfalvi
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics

Classification of Visualization Algorithms

- Indirect visualization: Intermediate representation is required
- Direct visualization: The discrete volume data is directly processed
  - Image-order approach – pixel-by-pixel processing
  - Object-order approach – voxel-by-voxel processing
  - Hybrid approach – combines the advantages of image-order and object-order techniques

Image-Order Approach – Ray Casting

- No analytically defined objects
- Surfaces are represented by a discrete implicit function
- A continuous reconstruction from the discrete representation is necessary (approximation/interpolation)
- The accurate surface normals are lost due to the discretization
- For the shading computations the normals have to be estimated (gradient filtering)

Display of Internal Structures

- Slicing: a cross-sectional slice of the volume is displayed by an appropriate color coding
- Isosurface rendering: reconstruction of opaque surfaces
- Semitransparent visualization: the different materials attenuate the emitted or reflected light differently

Ray Tracing vs. Ray Casting

- The classical ray tracing is recursive (models multiple reflections)
- In direct volume rendering usually just the primary rays are traced, since the reflected directions calculated from the estimated normals would be inaccurate resulting in an accumulated error
- If geometrical models are transformed to volumetric models by a signed distance transform, the gradients are accurate enough for a recursive volumetric ray tracing

Definition of the Ray Profile

- Volume data: Scalar field in the 3D space
\[ f(\mathbf{x}) \in \mathbb{R}^3, \mathbf{x} \in \mathbb{R}^3 \]
- Ray: half line
\[ \mathbf{r}(t) \in \mathbb{R}^3, t \in \mathbb{R}^1 > 0 \]
- Ray profile:
\[ f(\mathbf{r}(t)) \in \mathbb{R}^3, t \in \mathbb{R}^1 > 0 \]
Visual mapping of the ray profiles

- MIP
- Alpha-blending
- X-ray
- First-hit isorurface

First-hit Ray Casting

X-Ray Rendering

MIP - Maximum Intensity Projection

Accumulation - Alpha-blending

Two-level Volume Rendering

- Segmentation
- Different compositing operators are assigned to the different tissues
Style Transfer Functions

- Illustrative approach
- Different rendering styles are assigned to the different tissues

Recursive evaluation

- Back to front:
  \[ C_{\text{out}} = c(t_i) \cdot \alpha(t_i) + C_{\text{in}} \cdot (1 - \alpha(t_i)) \]
- Front to back:
  - opacity accumulation buffer is required
  - terminated if the accumulated opacity reaches a predefined threshold (early ray termination)

Classical Volume-Rendering Integral

- The parametric equation of the ray:
  \[ \mathbf{r}(t) = \mathbf{o} + \mathbf{d} \cdot t \]
- Volume-rendering integral:
  \[ I_{\text{max}} = \int_0^{t_{\text{max}}} I(t) e^{-\int_0^t \mu(s) ds} dt \]
- Discrete approximation:
  \[ I \approx \sum_{i=0}^N c(t_i) \cdot \alpha(t_i) \prod_{j=0}^{i-1} (1 - \alpha(t_j)) \]

Classical Ray Casting Pipeline

- Marc Levoy 1988
  1. \( C(i) \) and \( \alpha(i) \) are set by a transfer function
  2. Ray casting, interpolation
  3. Compositing

1. step: classification, shading

- Shading: \( f(i) \rightarrow C(i) \)
  - transfer function
  - Phong model
  - normal: estimated gradient
- Classification: \( f(i) \rightarrow \alpha(i) \)
  - Levoy 88: modulation by the gradient magnitude
  - enhances the well-defined transitions

2. step: Ray traversal

- regular sampling
- scan conversion
- voxel intersections
- equidistant samples
- all voxels used once
- samples weighted by lengths of ray segments
Ray traversal, interpolation

- Voxel-based vs. cell-based ray traversal
- Trilinear interpolation inside the 3D cells
- Bilinear interpolation on the 2D faces of the cells
- Trilinear interpolation:
  1. 4 new samples along the x-axis (interpolated square)
  2. 2 new samples along the y-axis (interpolated line segment)
  3. 1 new sample along the z-axis (interpolated point)
- Uniform or varying sampling distance: The compositing gives different results ⇒ opacity correction is needed

3. step: compositing

- Back-to-Front (B2F):
  - Out = ln_i(1 - α_i) + Cα_i, ln_i+1 = Out_i
  - example:
    - voxel i: C_i = red, α_i = 30%, ln_i = white
    - after the compositing: 70% white + 30% red
- Front-to-Back (F2B):
  - accumulated color: C_acc = C_acc + (1 - α_acc) C α_i
  - accumulated opacity: α_acc = α_acc + (1 - α_acc) α_i

Interpolation - resampling

Comparison of B2F to F2B

- Back-to-Front (B2F) for two voxels:
  - background: ln_i
  - 1. voxel: Out = ln_i(1 - α_i) + C α_i = ln_i
  - 2. voxel: Out = ln_i(1 - α_i) + C α_i = ln_i
  - ln_i(1 - α_i)(1 - α_i) + C α_i(1 - α_i) = C α_i
- Front-to-Back (F2B) for two voxels:
  - initialization: C_α = 0, α_α = 0
  - 2. voxel: C_α = C_α + (1 - α_α) C α_i = C α_i
  - 1. voxel: C_α = C α_i + (1 - α_i) C α_i
  - C_α = α_i + (1 - α_i) α_i
  - background: C_α = C α_i + (1 - α_i) C α_i(1 - α_i) = C α_i

Opacity-Weighted Color Interpolation

Interpolation without opacity weighting

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Interpolation with opacity weighting

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Pre-classification vs. Post-classification

- Pre-classification
  - The color and opacity values are assigned to the voxels
  - In the sample positions these values are interpolated (with opacity weighting)
- Post-classification
  - The densities and the gradients are interpolated
  - The color and opacity values are assigned to the sample points based on the transfer function
  - The sample points are shaded based on the interpolated gradients
**Pre-classification vs. Post-classification**

![Image](Pre-classification_vs_Post-classification.png)

**Transfer Functions**

- Input:
  - density
  - gradient (from the first-order derivatives)
  - curvature (from the second-order derivatives)
  - position (based on a segmentation mask)
- Output:
  - color (e.g. RGB vectors)
  - opacity

**Fuzzy Transfer Functions**

- Assumption: Each cell represents the mixture of at most two materials
- Linear combination of basis functions

![Image](Fuzzy_Transfer_Functions.png)

**Images generated by ray casting**

![Image](Images_generated_by_ray_casting.png)
**Curvature-Based Transfer Functions**

- Hessian matrix – the elements are the second-order derivatives
- The colors are determined from the eigenvalues of the Hessian matrix

\[
H = \begin{bmatrix}
\frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\
\frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\
\frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2}
\end{bmatrix}
\]

**Brute-force Ray Casting (B2F)**

```cpp
RGB Volume::RayCasting(Vector pos, Vector dir)
{
    RGB color_acc;
    DBL tmin = this->EntryPoint(), t = this->ExitPoint();
    while(t > tmin)
    {
        Vector sample = pos + dir * t;
        DBL density = this->Resample(sample);// trilinear interpolation
        Vector gradient = this->ResampleGradient();
        opacity = OpacityFunction(density, gradient, Magnitude());
        color = ColorFunction(density) * Shading(gradient);
        color_acc += (1 - opacity) + color * opacity;
        t++;
    }
    return color_acc;
}
```

**Brute-force Ray Casting (F2B)**

```cpp
RGB Volume::RayCasting(Vector pos, Vector dir)
{
    RGB color_acc; DBL opacity_acc = 0;
    DBL t = this->EntryPoint(), tmax = this->ExitPoint();
    while(t < tmax)
    {
        Vector sample = pos + dir * t;
        DBL density = this->Resample(sample);// trilinear interpolation
        Vector gradient = this->ResampleGradient();
        opacity = OpacityFunction(density, gradient, Magnitude());
        color = ColorFunction(density) * Shading(gradient);
        color_acc += (1 - opacity_acc) * color * opacity;
        opacity_acc += opacity_acc + opacity, t++;
    }
    return color_acc;
}
```

**First-Hit Ray Casting**

```cpp
RGB Volume::RayCasting(Vector pos, Vector dir, DBL threshold)
{
    DBL t = this->EntryPoint(), tmax = this->ExitPoint();
    while(t < tmax)
    {
        Vector sample = pos + dir * t;
        DBL density = this->Resample(sample); // trilinear interpolation
        if(density > threshold)
        {
            Vector gradient = this->ResampleGradient();
            color = Shading(gradient);
            return color;
        }
        t++;
    }
    return BACKGROUND;
}
```

**Acceleration techniques**

- Optimization of primitive operations with SIMD instructions (SSE)
- Utilization of coherence (data coherence, ray coherence frame coherence)
- Hierarchical data structures (pyramid, octree)
- Empty space leaping
- Application of potential fields
- Hybrid techniques (PARC – Polygon Assisted Ray Casting)
- GPU implementation
Vector class

```cpp
struct Vector {
    float x, y, z, w;
    Vector(float x0, float y0, float z0, float w0 = 0) { x = x0; y = y0; z = z0; w = w0; }
    Vector operator*(float a) { return Vector(x * a, y * a, z * a, w * a); }
    float Length() { return sqrt(x * x + y * y + z * z);  }
    Vector operator%(Vector &v) { return Vector(y*v.z-z*v.y, z*v.x - x*v.z, x*v.y-y*v.x); }
}
```

Vector class with SSE instructions

```cpp
struct Vector {
    float x, y, z, w;
    public:
        Vector operator+( Vector &v ) { _declspec(align(16)) Vector res;
            _asm {
                mov    esi, this
                movups xmm0, [esi]
                movups xmm1, [edi]
                movaps res, xmm0
            }
            return res;
        }
}
```

Early ray termination

```cpp
RGB Volume::RayCasting(Vector pos, Vector dir)
{
    RGB color_acc; DBL opacity_acc = 0;
    DBL t = this->GetSample(sample); // trilinear interpolation
    Vector gradient = this->GetSampleGradient();
    opacity = OpacityFunction(density, gradient.Magnitude());
    color = ColorFunction(density) * Shading(gradient);
    color_acc += (1 - opacity_acc) * color * opacity;
    opacity_acc += (1 - opacity_acc) * opacity, l++; //opacity_acc = 0.0001) break;
}
return color_acc;
```

Min/Max Octree

- For each block the minimal density \( d_{\text{min}} \) and the maximal density \( d_{\text{max}} \) are stored
- Threshold \( t \) defines the isosurface
- The largest block is searched for, where the given sample point is contained and \( t < d_{\text{min}} \) or \( t > d_{\text{max}} \)
- This block is not intersected by the isosurface for sure, so the next sample point is the exit point (empty space leaping)

Potential field

- For each cell the distance of the nearest surface point is stored
- This distance is added to the current sample point
- In between an intersection point cannot be missed

Hybrid acceleration

- Object-order methods:
  - the empty regions are easy to handle
  - relatively large overhead (projection, rasterization)
  - efficient for sparse data sets
- Image-order techniques:
  - the empty regions are difficult to handle
  - the rendering speed depends on the number of pixels rather than on the number of voxels
  - more efficient for uniformly distributed data
- Hybrid methods:
  - usually multi-pass techniques (combination of object-order and image-order passed)
  - combines the advantages of the two approaches
Those cells are searched for that are intersected by the isosurface.
The faces of these cells are rendered by the graphics hardware into two depth buffers (front buffer/back buffer).
Based on the depth buffers the entry end exit points of the rays are determined.

Object-order approach
The voxels are projected onto the screen one after another
The empty regions are not processed
Utilization of the cache coherence
Drawbacks
- The fine details are blurred
- Not the most appropriate technique for isosurface rendering
- The hidden regions are also processed

A voxel is not projected onto a single pixel, but leaves a footprint on several pixels.
The voxel value is distributed among the pixels of the footprint.
The larger the pixel/voxel ratio, the larger is the footprint, which blurs the image.

Footprint: integrated reconstruction kernel
If the reconstruction kernel is spherically symmetric, the footprint is circularly symmetric and, in case of parallel projection, independent from the viewing direction.
Perspective projection:
- The footprint is elliptical, but can be preintegrated
- The direction of elliptical footprints has to be taken into account

Support: a region covered by the 3D reconstruction kernel
To each voxel a kernel is associated that is weighted by the voxel value.
Each voxel leaves a 2D footprint on the screen (color, opacity).
The weighted footprints are accumulated on the screen.

Support: a region covered by the 3D reconstruction kernel
To each voxel a kernel is associated that is weighted by the voxel value.
Each voxel leaves a 2D footprint on the screen (color, opacity).
The weighted footprints are accumulated on the screen.
To each voxel a kernel is associated that is weighted by the voxel value. Each voxel leaves a 2D footprint on the screen (color, opacity). The weighted footprints are accumulated on the screen.

- The kernels belonging to the first slice are accumulated.
- The kernels belonging to the second slice are accumulated.

Splatting - advantages
- The splats can be preintegrated.
- Fast voxel projection.
- The interpolation is performed in the 2D screen space.
- In a special case (X-ray simulation) gives an analytical solution.
- Only the relevant voxels are processed (efficient for sparse volume data).
- Empty regions do not have to be encoded.

Splatting - implementation
- The voxels belonging to a slice are accumulated in a sheet buffer.
- The accumulation buffers are composited in F2B order.
- Those slices are processed that are mostly perpendicular to the viewing direction.

- The kernels belonging to the first slice are accumulated.
- The kernels belonging to the second slice are accumulated.
Splatting - implementation

- The accumulation buffer is composited into the frame buffer

![Diagram showing accumulation buffer and frame buffer](image)

Improving the reconstruction quality

- The distance between the resampling slices is decreased
- The reconstruction kernel is integrated in slabs accordingly

![Diagram showing slabs and kernels](image)

Calculation of the slabs

- The slabs are evaluated in a preprocessing

![Diagram showing slabs and kernels](image)

Splatting - implementation

1. Each voxel is transformed into the screen coordinate system (the z coordinate represents the distance from the image plane)
2. Bucket sorting based on the transformed z coordinate – to each accumulation buffer a bucket is associated
3. The occlusion map is initialized to zero opacity
4. For each accumulation buffer in F2B order
   For each footprint
     If the accumulated footprints are still translucent
       Rasterization of the footprint and compositing
     Refreshing of the occlusion map

Splatting – perspective projection

- In case of perspective projection, the footprint is elliptical
- Linear transformation with a resampling of a 2D precalculated footprint is required

![Diagram showing linear transformation](image)
Splatting with a 1D footprint

- Exploiting the symmetry, the footprint can be represented by a 1D array.
- The distance from the center has to be calculated (but the square root is a computationally expensive operation).
- The array is initialized such that it can be addressed by the square of the distance: $r_{x,y}^2 = (x - x_0)^2 + (y - y_0)^2$.
- During the rasterization, the squared distances are calculated incrementally: $r_{x+1,y}^2 = r_{x,y}^2 + 2(x - x_0) + 1$.

Elliptical splat with a 1D footprint

- The same 1D array is used as for the circular splats.
- The squared distance from the center is calculated from the equation of the ellipse: $r_{x,y}^2 = a(x - x_0)^2 + b(y - y_0)^2 + c(x - x_0)(y - y_0)$.
- The squared distance can be calculated incrementally: $r_{x+1,y}^2 = r_{x,y}^2 + 2a(x - x_0) + a + c(y - y_0)$.

Numerical error

- Object-order technique
- The shading is calculated for the voxels (preclassification).
- Interpolation with a spherically symmetric kernel (a trilinear kernel would not be efficient).
- Template-based voxel projection.
- The compositing is implemented with accumulation buffers.

Images generated by splatting

- Pre-classification vs. post-classification
**The reason of the blurring effect**
- The ideal low-pass filter is usually approximated by a Gaussian filter.
- Pre-classification: The colors are interpolated; therefore, due to the smoothing effect, the image gets blurred.
- Post-classification: The thresholding compensates the smoothing effect.

![Images showing original edge, blurred edge, and crisp edge](image1.png)

**Gradient estimation**
- The ideal derivative filter is the analytical derivative of the ideal sinc kernel.
- Derivative filtering with a Gaussian kernel:
  - \( H' \): analytical derivative of the Gaussian kernel.
  - \( DH \): Central differences combined with a Gaussian kernel.

![Graph showing derivative filters](image2.png)

**Splatting without blurring**
- Adaptation of the post-classification scheme.
- The accumulation buffers are used for the interpolation of the density values.
- The gradients are estimated from the accumulation buffers using central differences.
- A temporary storage of the accumulation buffers is necessary.
- The classification is performed on the accumulation buffers.

**Comparison**
- Pre-classification
- Post-classification

**Pixel-level operations**
- The post-classification scheme supports texture mapping and bump mapping.

![Comparison images](image3.png)
Hierarchical splatting

- Pyramidal structure (full octree without pointers)
- Each node stores the average density value of the corresponding cell
- The error of a cell is the squared deviation from its average
- During the rendering, the hierarchy is processed at such a level which guarantees that the error is below a predefined threshold
- The high-frequency details are represented by more nodes, while the homogeneous regions are represented by fewer nodes

Shear-Warp Factorization

- Hybrid technique – combines the advantages of the image-order and the object-order methods
- The projection is performed in the sheared space
- In the sheared space the slices are parallel to the image plane
- The resampling is simplified to a 2D operation (bilinear interpolation instead of a trilinear interpolation)
- The intermediate image is geometrically distorted; therefore, it needs to be corrected by an appropriate warp transformation

Shear-Warp Factorization

- Factorization of the viewing transformation:  
  \[ M_{view} = M_{view\to\text{shear}} \cdot M_{\text{shear}\to\text{warp}} \]
- The 3D shearing transformation:  
  \[ M_{\text{shear}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
- The 2D warp transformation is obtained from a system of linear equations:  
  \[ M_{\text{warp}2D} = M_{\text{shear}} \cdot M_{\text{warp}1D} \]
  \[ M_{\text{warp}1D} = \begin{bmatrix} 1 & -s_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
Ray templates

- In the sheared space each ray is evaluated based on the same template.
- The ray templates define the interpolation weights and the relative offsets of the voxels.

Run-Length Encoding

Images generated by the shear/warp algorithm