Footprint Evaluation for Volume Rendering

A Feed-forward Approach - a.k.a. Splatting

(We 'used' to be called Ohio Splatting University (OSU))

Process for volume rendering

- Reconstruct the continuous volume function
- Shade the continuous function
- Project this continuous function into image space
- Resample the function to get the image

Feed Backward vs. Feed Forward

- Feed Backward: Image space algorithm (Ray casting)
- Feed Forward: Object space method (Splatting)

Splatting (feed-forward)

Feed Backward vs. Feed Forward

Feed Forward: Splatting

Backward: ray casting

Fill the holes

We need to fill the pixel values between the volume projection samples

That is, to fit a continuous function through the discrete Samples

We can use convolution to do this
**Convolution**

Convolution:

\[ g(x,y) = \sum_{i} \sum_{j} f(i,j) h(x-i, y-j) \]

The output is a weighted average of inputs.

**Convolution (2)**

Another way of thinking convolution is to deposit each function value to its neighbor pixels.

**Volume Rendering and Convolution**

- Feed Backward (ray casting) views convolution as generating outputs as a weighted average of inputs.
- Feed Forward (splatting) views convolution as generating outputs as inputs distributing energy to outputs.

**3D Kernel for Splatting**

Need to know the 3D extent of each voxel, and then project the extent to the image plane.

**Footprint function**

\[ g(x,y,z) = \sum_{i} \sum_{j} \sum_{k} f(i,j,k) h(x-i, y-j, z-k) \]

Effect \([i,j,k] \rightarrow (x,y,z)\] = \(f(i,j,k) \times h(x-i, y-j, z-k)\)

Effect \([i,j,k] \rightarrow (x,y)\] = \(\int_{z}^{\infty} f(i,j,k) \times h(x-i, y-j, z-k) \, dz\)

Footprint Function (2)

Effect \([i,j,k] \rightarrow (x,y)\] = \(f(i,j,k) \int_{z}^{\infty} h(x-i, y-j, z-k) \, dz\)

This footprint function defines how much voxel \((i,j,k)\) will deposit its value to pixel \((i+x, j+y)\) = \(f(i,j,k) \times \text{footprint}(x,y)\).
Footprint Function (3)

Pixel \((i+x,j+y)\) receives \(f(i,j,k) \times \text{footprint}(x,y)\) value deposits

The final value of pixel \((i+x,j+y)\) will be a total sum of the contributions from its surrounding voxel projections.

Footprint Function (4)

\[
\text{footprint}(x,y) = \int h(x,y,z) \, dz
\]

- Evaluating \(\text{footprint}(x,y)\) on the fly is too time-consuming - involves integration of the kernel function \(h(x,y,z)\).
- We can build the footprint table at preprocessing time.
- The kernel function can be any (depends on the renderer).

Footprint Extent

Approximate the 3D kernel \((h(x,y,z))\) extent by a sphere.

Footprint Table

A popular kernel is a three-dimensional Gaussian.
As 1D integration of 3D Gaussian is still a 2D Gaussian - we can just skip the Z integration and evaluate the Gaussian function on 2D image space after voxel projection.

View-dependent footprint

It is possible to transform a sphere kernel into an ellipsoid.

- The projection of an ellipsoid is an ellipse.
- We need to transform the generic footprint table to the ellipse.

View-dependent footprint (2)

\[T^{-1}(x) = x'\]
Footprint Value Lookup

- For rectilinear meshes, the footprint of each sample is identical except for an image-space offset.
- The renderer only needs to calculate footprint function once for each view.
- Weight is calculated by table lookup at the footprint function value at each pixel that lies within the footprint’s extend.

Visibility

- Splatting uses the compositing operator to perform visibility
- Either front to back or back to front compositing (different formula)
- The problem for simply composite sample’s footprint onto the accumulation buffer sample by sample

Effects of No. of entries of the table

- Time versus space tradeoff
- If a lot of entries, nearest neighbor works fine
- If coarse, interpolate from nearest samples.
- For smaller table size, interpolation gives much better results.
- Images (figure 2 in the paper).

Results (1)

- The choice of kernel can affect the quality of the image.
- Examples of cone, gaussian, sync, and bilinear function.

Effects of Kernel function
Conclusion

- Different from “existing” algorithms (ray casting)
- More efficient (sometimes)
- Easy to make parallel