CSE 564: Visualization

Other Rendering Techniques: Points, Splats, Shear-Warp

Klaus Mueller
Stony Brook University
Computer Science Center for Visual Computing

X-Ray Rendering

- Estimate ray integral via discrete raycasting:

\[ p_i = \sum_j \sum_k v_j \cdot h(s_k, x(v_j)) \]

Complete discrete ray integral:

\[ s_k = \sum_j v_j \cdot h(X(s_k) - X(v_j)) \]

Reversing the order of \( j \) and \( k \):

\[ p_i = \sum_j v_j \sum_k h(s_k - x(v_j)) \]

X-Ray Point Splatting

- Example: projecting a volume of two points

1. rasterize footprint
2. add footprints
3. rasterize footprint
4. add footprints

\[ \tilde{h}(r_i) = \int_{-\infty}^{\infty} h(r_i, s) ds \]

Compute continuous ray integral at \( p_i \):

\[ p_i = \sum_j v_j \cdot \tilde{h}_j(r_i) \]
X-Ray Point Splatting

- Re-ordering was first recognized by Hanson and Wecksung for 2D CT (Hanson ’85)
  - Later independently discovered by Westover for 3D volume rendering (Westover ’89)
- Facilitates computation of the true ray integral
  - not just a discrete Riemann sum (raycasting)
- Pre-integrated footprint is stored into a table
  - Need a kernel function for which mappings into the footprint table can be defined for any orientation
  - The Gaussian is such a function

Point Projection

- Each point is represented by a 3D Gaussian $G_V$:
  $$G_V = \frac{1}{2\pi\rho^{0.5}} e^{-0.5(x-v_j)^T\rho^{-1}(x-v_j)}$$
  - $G_V$ is an ellipsoid to facilitate more general grids
  - It is a sphere for cubic grids
- A viewing matrix $M$ transforms $G_V$ into $G_{MVM}$:
  $$G_{MVM} = \frac{1}{|M^{-1}|} G_{MVM} (u - Mv_j - T)$$
  (Heckbert ’89, Zwicker ’01)

Point Projection

- Projection $P$ of $G_{MVM}$ is screen ellipse $P(G_{MVM})$
  - Find $v_j$’s screen projection $P(M \cdot v_j + T)$
  - Find linear mapping of $P(G_{MVM})$ into footprint table
  - Rasterize footprint table under $P(G_{MVM})$ at $P(V \cdot v_j)$

Blending

- Note: Gaussian kernels do not blend perfectly
  - A small ripple always remains:
    Typical range: (0.99845, 1.00249) (assuming a function of unity)
- The wider the Gaussians, the smaller the ripple
- In practice, a radius = 2.0 in volume space works well (given the appropriate Gaussian)
- See (Crawfis and Max, Vis ’93) for an optimized kernel
• Splatting seemingly reduces the interpolation complexity by one dimension:
  - Raycasting: interpolation of samples in 3D
  - Splatting: rasterization of footprints in 2D
• But…

- Consider magnification = 1
- Raycasting:
  - Commonly uses trilinear interpolation
  - Requires 8 points to calculate one ray sample point
  - Total complexity: $O(8 \cdot n^3)$
- Splatting:
  - Uses Gaussian kernel of radius=2
  - Footprint rasterization touches 16 pixels
  - Total complexity: $O(16 \cdot n^3)$

• Does this mean that raycasting is more efficient than splatting?
• It depends….
  - Spatially intricate objects are good candidates for point-based rendering (splatting)
  - But the simplicity of splatting has advantages even for less favorable objects

• Generally, only need to store relevant points
  - Non-air points, masked-out points, ROI-points
• Provides easy space-leaping for irregular objects
• Storage schemes (in increasing order of spatial coherence):
  - List of points, sorted by value (fast iso-contouring)
  - RLE list of points (fast transformations and sparse)
  - Octree with hierarchical bins of points

Storage Complexity
Rendering

- RLE list facilitates fast incremental arithmetic for point projection in software
- Texture mapping hardware can also be used
  - Texture map footprint onto a square polygon
  - Set GL blending functions, etc.
  - Warp polygon according to point’s screen space ellipse
  - Align the warped polygon with the screen
  - Project polygon to the screen

Aliasing

- In perspective or at low magnifications, some volume portions may be sampled below Nyquist
  
  \[
  \text{Ray grid sampling rate} > \text{volume grid sampling rate} \quad \rightarrow \text{no aliasing}
  \]
  
  \[
  \text{Ray grid sampling rate} \leq \text{volume grid sampling rate} \quad \rightarrow \text{aliasing}
  \]

Aliasing

- Effects of aliasing
  
  - checkerboard tunnel
  - terrain

Anti-Aliasing

- Adapt kernel bandwidth for proper anti-aliasing
- Amounts to a stretch of the 3D kernel

  (Swan ‘97, Mueller ‘98)
Anti-Aliasing

- Conveniently done in perspective (ray-) space

Anti-Aliasing

- Compute the Gaussian ellipsoid in ray space
  - Calculate the Jacobian $J$ of the local perspective distortion (varies for each point)
  - Compute the ray space ellipsoid $G_{JMV}$ using $J$

\[
G_{MV} = \frac{1}{M^2} G_{MVJ} (u - Mv_j - T)
\]

\[
G_{JM} = \frac{1}{|J^{-1}|} G_{JMVM} (x - x_k)
\]

Anti-Aliasing - Results

Compositing - Raycasting

- Reconstruction followed by compositing

\[
c = C(s_k) \cdot \alpha(s_k) \cdot (1 - \alpha) + c
\]

\[
\alpha = \alpha(s_k) \cdot (1 - \alpha) + \alpha
\]
Compositing - Splatting

- Reconstruction not separable from compositing

\[
\tilde{h}(r) = \int_{-\text{ext}}^{\text{ext}} h(r,s)ds
\]

\[
c = C(v_j) \cdot \tilde{h}(r_j) \cdot \alpha(v_j) \cdot \tilde{h}(r) \cdot (1 - \alpha) + c
\]

\[
\alpha = \alpha(v_j) \cdot \tilde{h}(r_j) \cdot (1 - \alpha) + \alpha
\]

Compositing

- Two strategies devised by Westover (Westover '89, '90)

- Composite every point:
  - Shown in previous slide
  - Fast and simple
  - Leads to “sparkling” in animated viewing

- Axis-aligned sheet-buffers:
  - Add splats within sheets most parallel to image plane
  - Composite these sheets in depth-order
  - Leads to “popping” artifacts in animated viewing

Axis-Aligned Sheet-Buffers

- Eliminates popping
  - Slicing slab cuts kernels into sections
  - Kernel sections are added into sheet-buffer
  - Sheet-buffers are composited

Image-Aligned Sheet-Buffers

- Binary cube

(Mueller '98)
Image-Aligned Sheet-Buffers

- Footprint mapping as usual
  - Requires multiple footprint rasterizations per point

Pre-Classified Splatting

- Original edge
- Sampled edge
- Classification and shading
- Splatted with Gaussian kernel
- Reconstruction: blurred edge image

One Solution: Edge Splats

- Edge splats (Huang ‘98)
  - replace normal splat by special edge splat

- Shortcomings:
  - pre-processing required
  - problems with discontinuities
  - “micro-edges” are hard to resolve
Pre-Classified Rendering

Rendering Loop

- Color and opacity volume
- Classify and shade
- Raw density volume
- Viewing parameters

Splat into sheet-buffer ➔ Composite sheet-buffer ➔ Advance sheet-buffer ➔ Image

Post-Classified Rendering

Rendering Loop

- Splat into sheet-buffer ➔ Classify and shade ➔ Composite sheet-buffer ➔ Advance sheet-buffer ➔ Image

Note: this can only be done with image-aligned sheet buffers (Mueller ‘99)

Post-Classified Splatting

- Original edge
- Sampled edge
- Splatted with Gaussian kernel
- Reconstruction: blurred edge
- Classification: crisp edge image

Pre-shaded ➔ post-shaded, central difference ➔ post-shaded, gradient splats

Sheet buffers: current, current-1, current+1, current
Post-Classified Splatting

pre-shaded

post-shaded

Perspective Pre-Warp Revisited

Camera space

Ray space

EWA Volume Splatting

- Formally separates volume space (pre-) filtering from screen space (post-) filtering

EWA (Elliptical Weighted Average) volume resampling filter

Analysis

Warped reconstruction kernel

Low-pass filter

Resampling filter

Minification

Magnification
Effects

From Volume To Surface

- Iso-surface of a volumetric point-based object is represented by a hull of Gaussian kernels.
- Flattening the points in direction of the surface normals yields a more exact representation.
- In the limit get a surface composed of 2D Gaussians (aka surface points or surfels).

Volumetric vs. Surface Points

- Volumetric points:
  - Most often on a regular lattice
  - Represent both boundary and interior
  - Overlapping points reconstruct volumetric object
  - Different iso-surfaces, shapes, and compositions can be produced via transfer functions on the fly
- Surface points:
  - Irregular distribution (on the surface)
  - Usually located only on boundaries

Algorithms: Surfaces

- Surfels and Surface Splatting (Zwicker et al.)
- QSplat (Rusinkiewicz et al.)

- Main point 😊
  - points can represent fine detail better than triangles
**Primitives: Surfels and QSplats**

- Traverse point hierarchy from top to bottom
  - For each block of points, find level where the local resolution of the projected point set matches the screen resolution (oversample for better quality)
- Splat the selected points into the z-Buffer
- Blend visible points, fill holes, shade

**Algorithms: Volumes**

- Rymon-Lipinski et al., Tricoche et al. (Vis ‘04)
- Vega Higuera (Vis ‘05)

**Emphasis:**
- iso-surface rendering (obviously)
- fast data traversal and point localization (use span-triangle)
- How about mixing surface points and volume points?

**Alternative Grids**

- Body-centered cartesian (BCC) grid:
  - Reduces # of required point samples to 70.3%
  - 4D BCC grid requires only 50% of the equivalent 4D cubic grid samples
Alternative Grids

- Notes:
  - BCC grids assume spherically bandlimited signal
  - Under that assumption compression is lossless

- Rendering (Theussl ‘01):
  - All usual point rendering methods are applicable
  - Need to shift slices by $1/\sqrt{2}$

- Turbulent Jet 4D CC
  - 99 time steps (168M)
  - Relevant voxels: 9.4M
  - 3D extracted: 127k
  - Size RLE list: 146k
  - Render time: 1.23s

- Turbulent Jet 4D BCC
  - 138 time steps (87M)
  - Relevant voxels: 7.4M
  - 3D extracted: 107k
  - Size RLE list: 146k
  - Render time: 1.01s (71%)  
  (Neophytou ‘02)

Alternative Grids

- Animations of time-varying datasets:
  - turbulent jet
  - turbulent flow

Detail Modeling

- Footprints do not have to serve interpolation alone (via the pre-integrated kernel function)
- They can be used to add additional detail or information between the sample points
- The Gaussian footprint provides the blending

(vector field splat)

(Crawfis/Max ‘93)
Shear-Warp Nuts and Bolts

\[ \mathbf{M}_{\text{view}} = \mathbf{M}_{\text{shear}} \cdot \mathbf{M}_{\text{warp}} \]
Shear-Warp Nuts and Bolts

- Cheap bilinear interpolation within slice
- Keep baseplane resolution at volume resolution
  - Allows re-use of pre-computed bilinear weights

Shear-Warp Nuts and Bolts

- Each voxel stores index into reflectance map
- For each slice: Compute bilinear weights
- For each voxel square:
  - Shade voxels using the reflectance cube
  - Interpolate shaded voxels and composite result

Shear-Warp Nuts and Bolts

- RLE encoding of voxel and pixel runs
  - skip over transparent voxels quickly
  - skip over opaque pixels quickly

Shear-Warp Nuts and Bolts

- RLE encoding of voxel and pixel runs
  - skip over transparent voxels quickly
  - skip over opaque pixel quickly
• RLE encoding of voxel and pixel runs
  - skip over transparent voxels quickly
  - skip over opaque pixel quickly

• RLE encoding of voxel and pixel runs
  - skip over transparent voxels quickly
  - skip over opaque pixel quickly
Shear-Warp Nuts and Bolts

- Warping the baseplane image into screen image is a cheap 2D operation

Sources of Speed

- Cheap bilinear interpolation
- Use of pre-computed (per slice) bilinear weights
- Fixed number of rays
  - Zooming is deferred to 2D warp step
- Very efficient occlusion culling

Drawbacks

- Ray sample distance is view dependent
  - Slice artifacts may appear at oblique view angles
- For zooms the baseplane image is magnified
  - Causes blur
- Interpolation of shaded voxels (pre-shading)
  - Also causes blur
- 3 axis-aligned RLE lists are used
  - Memory consumption is tripled
  - Popping may occur on list change

Rendering Results

Standard (0.30s) 1 interm. slice (0.40s) + matched sampl. (3.3s)

Engine (128^3) + post-class (2.9s)
Rendering Results

- Engine (enlarged detail)

standard  one intermediate slice  + matched sampling  + post-class