Correct resolution rendering of trimmed spline surfaces

Ruijin Wu and Jörg Peters

University of Florida

SPM 2014
Capture shape w/o considering other surfaces
Industrial Workflow: Overfit + Trim

- Capture shape w/o considering other surfaces
- Trim the surfaces back to match constraints
Benefits

Practice
Benefits

Practice

- **Accuracy** currently: predefined triangulation level
Benefits

Practice

- **Accuracy** currently: predefined triangulation level
- **Latency** adjust trim, modeler recomputes triangles
Benefits

Practice

- **Accuracy** currently: predefined triangulation level
- **Latency** adjust trim, modeler recomputes triangles

Theory
Practice

- **Accuracy** currently: predefined triangulation level
- **Latency** adjust trim, modeler recomputes triangles

Theory

- **Accuracy** domain $\leftrightarrow$ projected image resolution
Benefits

Practice
- **Accuracy** currently: predefined triangulation level
- **Latency** adjust trim, modeler recomputes triangles

Theory
- **Accuracy** domain $\leftrightarrow$ projected image resolution
- **Latency** Lean parallel data structures
Goal

- High precision rendering
- Interactive frame rate
Surface and Trim Curve

- Surface piece $S$ maps the unit rectangle to 3-space.

$\begin{pmatrix} u \\ v \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad \rightarrow \quad S$
Surface and Trim Curve

- Surface piece $S$ maps the unit rectangle to 3-space.
- Trim curves define and restrict the domain of the surface.
Surface piece $S$ maps the unit rectangle to 3-space.

Trim curves define and restrict the domain of the surface.

“inside” is determined by ray test in the $uv$ domain.
1 Basics: Rendering Trimmed Surfaces

2 Earlier approaches

3 Correct resolution trimming

4 Data structure for maximally Fat Correct Scan Lines

5 Leveraging Correct Tessellation and the Graphics Pipeline

6 Comparisons

7 Summary
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix} u \\ v \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}
\]
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
  u \\
  v
\end{pmatrix} \rightarrow \begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\rightarrow \begin{pmatrix}
  \tilde{x} \\
  \tilde{y}
\end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its \(uv\).
Ray trace or trim texture?

Map in from \((u, v)\) to \((\hat{x}, \hat{y})\):

\[
\begin{bmatrix}
\begin{array}{l}
u
\end{array}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\begin{array}{l}
x
\end{array}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\begin{array}{l}
\hat{x}
\end{array}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\begin{array}{l}
y
\end{array}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\begin{array}{l}
\hat{y}
\end{array}
\end{bmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its uv.

- ray trace (in uv domain) [e.g. Pabst06, Schollmeyer09]
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
u \\
v
\end{pmatrix}
\rightarrow
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\tilde{x} \\
\tilde{y}
\end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its \(uv\).

- ray trace (in \(uv\) domain) [e.g. Pabst06, Schollmeyer09]
  + precise
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
    u \\
v
\end{pmatrix}
\rightarrow
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\tilde{x} \\
\tilde{y}
\end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its \(uv\).

- ray trace (in \(uv\) domain) [e.g. Pabst06, Schollmeyer09]
  + precise - several trim curves, many curve segments
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
u \\
v
\end{pmatrix} \rightarrow \begin{pmatrix}x \\ y \\ z\end{pmatrix} \rightarrow \begin{pmatrix}x \\ y \\ z\end{pmatrix} \rightarrow \begin{pmatrix}\tilde{x} \\ \tilde{y}\end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knowns its \(uv\).

- ray trace (in uv domain) [e.g. Pabst06, Schollmeyer09]
  + precise - several trim curves, many curve segments
  - numerical stability (multiple roots, curves not implicit) [LB05]
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
u \\ v
\end{pmatrix} \rightarrow \begin{pmatrix}
x \\ y \\ z
\end{pmatrix} \rightarrow \begin{pmatrix}
x \\ y \\ z
\end{pmatrix} \rightarrow \begin{pmatrix}
\tilde{x} \\ \tilde{y}
\end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its \(uv\).

- ray trace (in uv domain) [e.g. Pabst06, Schollmeyer09]
  - precise
  - several trim curves, many curve segments
  - numerical stability (multiple roots, curves not implicit) [LB05]
- in/out ‘trim texture’ (uv grid) [e.g. Guthe05]
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
u \\
v
\end{pmatrix}
\rightarrow
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
\rightarrow
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\tilde{x} \\
\tilde{y}
\end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its uv.

- ray trace (in uv domain) \([e.g. \ Pabst06, \ Schollmeyer09]\)
  - precise  - several trim curves, many curve segments
  - numerical stability (multiple roots, curves not implicit) \([LB05]\)
- in/out ‘trim texture’ (uv grid) \([e.g. \ Guthe05]\)
  - fast lookup
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
u \\
v
\end{pmatrix} \rightarrow
\begin{pmatrix}x \\ y \\ z
\end{pmatrix}
\rightarrow
\begin{pmatrix}x \\ y \\ z
\end{pmatrix}
\rightarrow
\begin{pmatrix}\tilde{x} \\ \tilde{y}
\end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its \(uv\).

- ray trace (in \(uv\) domain) [e.g. Pabst06, Schollmeyer09]
  + precise  - several trim curves, many curve segments
  - numerical stability (multiple roots, curves not implicit) [LB05]
- in/out ‘trim texture’ (uv grid) [e.g. Guthe05]
  + fast lookup + robust
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
u
\end{pmatrix} \rightarrow \begin{pmatrix}x \\ y \\ z\end{pmatrix} \rightarrow \begin{pmatrix}x \\ y \\tilde{y}\end{pmatrix} \rightarrow \begin{pmatrix}\tilde{x} \\ \tilde{y}\end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its \(uv\).

- ray trace (in \(uv\) domain) [e.g. Pabst06, Schollmeyer09]
  - precise
  - several trim curves, many curve segments
  - numerical stability (multiple roots, curves not implicit) [LB05]
- in/out ‘trim texture’ (uv grid) [e.g. Guthe05]
  - fast lookup + robust
  - resolution
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix} u \\ v \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its \(uv\).

- ray trace (in \(uv\) domain) [e.g. Pabst06, Schollmeyer09]
  - precise  - several trim curves, many curve segments
  - numerical stability (multiple roots, curves not implicit) [LB05]
- in/out ‘trim texture’ (uv grid) [e.g. Guthe05]
  - fast lookup  + robust
  - resolution  - (re)computation (parallel write),
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{bmatrix}
u \\ v
\end{bmatrix}
\rightarrow
\begin{bmatrix}
x \\ y \\ z
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\tilde{x} \\ \tilde{y}
\end{bmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its uv.

- ray trace (in uv domain) [e.g. Pabst06, Schollmeyer09]
  + precise - several trim curves, many curve segments
  - numerical stability (multiple roots, curves not implicit) [LB05]
- in/out ‘trim texture’ (uv grid) [e.g. Guthe05]
  + fast lookup + robust
  - resolution - (re)computation (parallel write), - space
Ray trace or trim texture?

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
    u \\
    v
\end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}
\]

Assume: Each pixel \(P(x(u,v))\) knows its \(uv\).

- ray trace (in \(uv\) domain) [e.g. Pabst06, Schollmeyer09]
  + precise  - several trim curves, many curve segments
  - numerical stability (multiple roots, curves not implicit) [LB05]

- in/out ‘trim texture’ (\(uv\) grid) [e.g. Guthe05]
  + fast lookup + robust
  - resolution  - (re)computation (parallel write),  - space

- hybrid = robust + fast + sufficient precision?
Outline

1. Basics: Rendering Trimmed Surfaces
2. Earlier approaches
3. Correct resolution trimming
4. Data structure for maximally Fat Correct Scan Lines
5. Leveraging Correct Tessellation and the Graphics Pipeline
6. Comparisons
7. Summary
Surface piece $S$ maps the unit rectangle to 3-space.
Trim curves define and restrict the domain of the surface.
“inside” is determined by ray test *in the uv domain.*
‘fat’ rays – how fat?
Surface piece $S$ maps the unit rectangle to 3-space.
Trim curves define and restrict the domain of the surface.
“inside” is determined by ray test in the $uv$ domain.
‘fat’ rays – how fat? ‘simplify’ trim curve – how simple?
Predicting correct resolution

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix} u \\ v \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}
\]
Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix} u \\ v \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}
\]

‘pull back’ pixel grid need distinct pre-images
Predicting correct resolution

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix} u \\ v \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}
\]

‘pull back’ pixel grid need **distinct pre-images** avoid non-linear grid
Predicting correct resolution

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix}
  u \\
  v
\end{pmatrix} \rightarrow \begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix} \rightarrow \begin{pmatrix}
  \tilde{x} \\
  \tilde{y}
\end{pmatrix}
\]

‘pull back’ pixel grid need distinct pre-images avoid non-linear grid

Correct Resolution
Math 101: Determine the $v$-scan line density

The screen space distance between two $v$-scan lines is:

$$|\tilde{x}(u, v) - \tilde{x}(u, v + h)| = h|\tilde{x}_v(u, v^*)|, \quad \tilde{x}_v := \frac{\partial \tilde{x}}{\partial v}.$$
Math 101: Determine the $v$-scan line density

The screen space distance between two $v$-scan lines is:

$$|\tilde{x}(u, v) - \tilde{x}(u, v + h)| = h |\tilde{x}_v(u, v^*)|, \quad \tilde{x}_v := \frac{\partial \tilde{x}}{\partial v}.$$ 

If, for all $v \in V_i$ and the $v$-scan line spacing $h > 0$,

$$h \rho_i(v) < 1,$$

$$\rho_i(v) := \max \{ \sup_u |\tilde{x}_v(u, v)|, \sup_u |\tilde{y}_v(u, v)| \},$$
Math 101: Determine the \( \nu \)-scan line density

The screen space distance between two \( \nu \)-scan lines is:

\[
|\tilde{x}(u, \nu) - \tilde{x}(u, \nu + h)| = h |\tilde{x}_\nu(u, \nu^*)|,
\]
\[
\tilde{x}_\nu := \frac{\partial \tilde{x}}{\partial \nu}.
\]

If, for all \( \nu \in V_i \) and the \( \nu \)-scan line spacing \( h > 0 \),

\[
h \rho_i(\nu) < 1,
\]
\[
\rho_i(\nu) := \max\{\sup_u |\tilde{x}_\nu(u, \nu)|, \sup_u |\tilde{y}_\nu(u, \nu)|\},
\]

then the \( \tilde{x} \)-distance between the screen images of the two \( \nu \)-scan lines \( \tilde{x}(u, \nu_j) \) and \( \tilde{x}(u, \nu_j + h) \) is less than a pixel and so is the \( \tilde{y} \)-distance.
Summary: Predicting correct resolution

Map in from \((u, v)\) to \((\tilde{x}, \tilde{y})\):

\[
\begin{pmatrix} u \\ v \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}
\]

Correct Resolution

\[
h < 1/\rho_i(v) \quad \rho_i(v) := \max\{\sup_u |\tilde{x}_v(u, v)|, \sup_u |\tilde{y}_v(u, v)|\}
\]

\(v\)-scan line spacing
Outline

1. Basics: Rendering Trimmed Surfaces
2. Earlier approaches
3. Correct resolution trimming
4. Data structure for maximally Fat Correct Scan Lines
5. Leveraging Correct Tessellation and the Graphics Pipeline
6. Comparisons
7. Summary
Uniformly partition the domain into $n_V$ $v$-strips. (First level)
Uniformly partition the domain into \( n_V \) \( \nu \)-strips. (First level)

Uniformly partition each \( \nu \)-strip into its own number of \( \nu \)-scan lines that guarantees correct resolution. (Second level)
Two-level scanline hierarchy

- Uniformly partition the domain into $n_V$ $v$-strips. (First level)
- Uniformly partition each $v$-strip into its own number of $v$-scan lines that guarantees correct resolution. (Second level)
- Store the $u$-coordinate of intersections for each $v$-scan line.
Uniformly partition the domain into $n_V$ $v$-strips. (First level)

Uniformly partition each $v$-strip into its own number of $v$-scan lines that guarantees correct resolution. (Second level)

Store the $u$-coordinate of intersections for each $v$-scan line.

Trim test: Look up position of each pixel’s pre-image in the table.
Outline

1. Basics: Rendering Trimmed Surfaces
2. Earlier approaches
3. Correct resolution trimming
4. Data structure for maximally Fat Correct Scan Lines
5. Leveraging Correct Tessellation and the Graphics Pipeline
6. Comparisons
7. Summary
Tessellation of curve and surface

- The trim curve is tessellated into line segments.

Subdividable linear efficient function envelopes = SLEFE
A tight bound of the deviation between the curve/surface and its piecewise linear approximation [Yeo et al 2012].
Tessellation of curve and surface

- The trim curve is tessellated into line segments.
- The surface is tessellated into triangles.

Subdividable linear efficient function envelopes = SLEFE

A tight bound of the deviation between the curve/surface and its piecewise linear approximation [Yeo et al 2012].
Using the GPU pipeline

Trim curves

$p_j$ per pixel (9) (previous frame)

Calculate $\rho_j$ per pixel per pixel

Compute $\mu_i$, base $i$, $\mu$ per pixel

Tessellate curves and intersect with scanline

Sort intersections mark in or out

Intersection Table

Compute Pass

Tessellation Engine

Geometry Shader

Fragment Shader

Patches

Tessellate Patches

Calculate per triangle $x_v, y_v$

in/out test Compute $\rho_j$

$\rho_j$ per pixel (9) (next frame)
Using the GPU pipeline

- iPass algorithm [Yeo et al. 2012] for surface rendering
  https://bitbucket.org/surflab/ipass_gl4
Using the GPU pipeline

- iPass algorithm [Yeo et al. 2012] for surface rendering
- Calculate ν-scan line spacing, $\rho_j$ per pixel
Using the GPU pipeline

- iPass algorithm [Yeo et al. 2012] for surface rendering
- Calculate $\nu$-scan line spacing, $\rho_j$ per pixel
- in/out test per pixel
Using the GPU pipeline

- iPPass algorithm [Yeo et al. 2012] for surface rendering
- Calculate \( \nu \) - scan line spacing, \( \rho_j \) per pixel
- in/out test per pixel
- Build \( u \) - intercept table
1. Basics: Rendering Trimmed Surfaces
2. Earlier approaches
3. Correct resolution trimming
4. Data structure for maximally Fat Correct Scan Lines
5. Leveraging Correct Tessellation and the Graphics Pipeline
6. Comparisons
7. Summary
Performance

Real-time interactive frame rate.

(a) Performance at different zoom levels.  (b) Performance at different screen resolution.

R Wu, J Peters (U Florida)  Correct Resolution Trimmed Surfaces  SPM 2014, HK 14 / 16
GPU memory usage

Compare to texture based technique

![Graph showing GPU memory usage with different camera positions. The graph compares the size (MB) of texture and u-intercept table memory usage. The graph shows a significant increase in memory usage for the texture method as camera position changes, while the u-intercept table method remains constant.]
Outline

1. Basics: Rendering Trimmed Surfaces
2. Earlier approaches
3. Correct resolution trimming
4. Data structure for maximally Fat Correct Scan Lines
5. Leveraging Correct Tessellation and the Graphics Pipeline
6. Comparisons
7. Summary
Contributions

- Determine a *provably* optimal scan density for trim test.
Contributions

- Determine a *provably* optimal scan density for trim test.
- Efficient *semi-uniform* scan-line data structure.
Contributions

- Determine a \textit{provably} optimal scan density for trim test.
- Efficient \textit{semi-uniform} scan-line data structure.
- GPU friendly: real-time trim curve editing with simple add-on.
Contributions

- Determine a *provably* optimal scan density for trim test.
- Efficient *semi-uniform* scan-line data structure.
- GPU friendly: real-time trim curve editing with simple add-on.

Questions?