Massively-Parallel Vector Graphics

Francisco Ganacim    Rodolfo S. Lima
Luiz Henrique de Figueiredo    Diego Nehab
IMPA

ACM Transactions on Graphics
(Proceedings of ACM SIGGRAPH Asia 2014)
Vector graphics are everywhere

clip-paths to the shortcut tree like any other path geometry, and maintain in each shortcut tree cell a stream that matches the scene grammar described in section 3. Clipping operations are performed per sample and with object precision.

When evaluating the color of each sample, the decision of whether or not to blend the paint of a filled path is based on a Boolean expression that involves the results of the inside-outside tests for the path and all currently active clip-paths. Since this expression can be arbitrarily nested, its evaluation seems to require one independent stack per sample (or recursion). This is undesirable in code that runs on GPUs. Fortunately, as discussed in section 4.3, certain conditions (see the pruning rules) allow us to skip the evaluation of large parts of the scene. These conditions are closely related to the short-circuit evaluation of Boolean expressions. Once we include these optimizations, it becomes apparent that the value at the top of the stack is never referenced. The successive simplifications that come from this key observation lead to the flat clipping algorithm, which does not require a stack (or recursion).

**Flat clipping** The intuition is that, during a union operation, the first inside-outside test that succeeds allows the algorithm to skip all remaining tests at that nesting level. The same happens during an intersection when the first failed inside-outside test is found. Values on the stack can therefore be replaced by knowledge of whether or not we are currently skipping the tests, and where to stop skipping. The required context can be maintained with a finite-state machine.

![Diagram](image)

**Figure 12:** State transition diagram for the finite-state machine of the flat-clipping algorithm.

Two transitions away from S. The first transition happens when an activate operation is found. Looking at the scene grammar, we see that this can only happen if the machine arrived at S due to a c1 transition from P. In other words, an entire clip-path test has succeeded, and therefore we transition unconditionally back to P.

The second transition happens when a matching c1 is found. The condition u = 0 means the machine is not inside a nested clip-path test, so it simply transitions back to P. If the machine is skipping inside a nested clip-path test, one of the inner clip tests must have passed, and therefore the outer test can be short-circuited as well. The machine simply resets the stop depth to the outer level and continues in state S.

The remaining transitions are between P and SA. If the machine finds a 1 while in state P, it must have been performing a clip-path test that continued unconditionally to S. The machine then transitions back to S. If the machine finds a 1 while in state SA, it must have been performing a clip-path test that continued unconditionally to S, but did not yield a new clip-path test.

The machine then transitions back to S.

The machine then transitions back to S.

The machine then transitions back to S.
Vector graphics are everywhere
Vector graphics are everywhere
Points to be made

• 2D graphics incredibly prevalent
• 2D graphics is not a “solved problem”
• It deserves more attention
• Can benefit from parallelism
  • Increased computational power
• Needs new algorithms
Diffusion-based vector graphics

[Orzan et al. 2008] [Finch et al. 2011] [Sun et al. 2012 and 2014]
Related work

PATH-BASED VECTOR GRAPHICS
Basic concepts are *paths* and *paints*
Paths

Closed contours
Inside-outside test

Winding numbers

Even-odd rule
Inside-outside test

Winding numbers

Non-zero rule
Paints

Radial gradient

Linear gradient

Texture
Availability

• Formats & languages
  • PostScript, CDR, PDF, SVG, OpenXPS, AI
  • TTF fonts, Type 1 fonts

• Editors
  • Adobe Illustrator, CorelDraw, Inkscape, FontForge, ...

• Rendering tools & APIs
  • NV_Path_Rendering, OpenVG, Cairo, Qt, MuPDF, GhostScript, Apple’s, Adobe’s, Microsoft’s, ...
Rasterization or rendering

Generate image at chosen resolution for display or printing
Traditional rendering algorithm

• Render one shape after the other

  for all shapes
  prepare for acceleration
  for all samples in shape
  blend paint over output

• Most tools follow this approach
Active-edge-list polygon filling

- Uses spatial coherence in horizontal spans

[Wylie et al. 1967]
Stencil-based polygon filling

• Rasterize winding numbers into stencil

[Neider et al. 1993]
Curve rendering by graphics hardware

- Constrained triangulation + affine implicitization

[Loop & Blinn 2005]
Implicitization

Theorem: A polynomial parametric curve

\[ c(t) = (x(t), y(t)) \]

has a polynomial implicit form \( C(x, y) \) with

\[ C(x_p, y_p) = 0 \iff \exists t_p \mid c(t_p) = (x_p, y_p) \]

• Different methods

  • Sederberg [1984]
    • Based on Cayley-Bézout or Sylvester
  • Loop & Blinn [2005]
    • Based on Salmon (affine implicitization)

\[
c(t) = \begin{bmatrix} -3 \\ -1 \end{bmatrix} (1 - t)^2 + \begin{bmatrix} 0 \\ 2 \end{bmatrix} 2t(1 - t) + \begin{bmatrix} 3 \\ -1 \end{bmatrix} t^2 \iff C(x, y) = x^2 + 6y - 3
\]
NV_Path_Rendering

• Stencil-based filling with affine implicitization
• Complete, state-of-the-art pipeline

[Neider et al. 1993] + [Loop & Blinn 2005] ≈ [Kokojima et al. 2006] ≈ [Kilgard & Bolz 2012]
Alternative approach
Magnification with image textures

• Can become blurry at high magnification levels

[Nehab & Hoppe 2008]
Magnification with vector textures

• Maintains sharpness indefinitely

[Nehab & Hoppe 2008]
General warps in object space

[Nehab & Hoppe 2008]
Vector texture rendering algorithm

• For texture mapping and effects
  for all shapes
    insert into acceleration structure
  for all samples
    for subset of shapes containing sample
      blend paint into output

• Mostly limited to academia
  [Sen 2004] [Ramanarayanan et al. 2004] [Qin et al. 2008]
  [Parilov & Zorin 2008] [Nehab & Hoppe 2008]
Comparison of rendering algorithms

Vector textures
• Extensive pre-processing
• Retained mode
• Samples are independent
• General warps
• Analogous to Ray-tracing

Traditional
• Modest preprocessing
• Immediate mode
• Sample cost is amortized
• Limited warps
• Analogous to Z-buffering
State of the art in accelerated rendering

<table>
<thead>
<tr>
<th>Vector textures</th>
<th>Traditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>for all shapes</td>
<td>for all shapes</td>
</tr>
<tr>
<td>insert in acceleration structure</td>
<td>prepare for acceleration</td>
</tr>
<tr>
<td>for all output samples</td>
<td>for all shapes</td>
</tr>
<tr>
<td>for subset of shapes covering sample</td>
<td>for all shape samples</td>
</tr>
<tr>
<td>blend paint into output</td>
<td>blend paint into output</td>
</tr>
</tbody>
</table>

[Nehab & Hoppe 2008] [Kilgard & Bolz 2012] (NV_Path_Rendering)
Massively-Parallel Vector Graphics

Goal

for all segments of all shapes
insert in acceleration structure

for all output samples
for subset of shapes covering sample
blend paint into output

Ours [Ganacim et al. 2014]
Contributions

• New primitive: *Abstract segment*
  • Based on implicitization, no intersection computations

• New acceleration data structure: *The Shortcut Tree*
  • Optimal, adaptive, segment-parallel construction

• State-of-the-art rendering quality
  • No compromises
Finding the right primitive

ABSTRACT SEGMENTS
Does shape cover sample?
Does ray intersect with segment?
Computing intersections

• Segment is \( c(t) = (x(t), y(t)), \quad t \in [0, 1] \)
• Sample at \( (x_s, y_s) \)

• Intersection test
  • Solve \( y(t) = y_s \) for \( t \)
  • For each \( t_i \in [0, 1] \) such that \( x(t_i) > x_s \)
  • Test sign of \( y'(t_i) \) to inc/dec winding number

• Requires solving quadratics and cubic equations
  • Complicated, slow, not robust
Monotonic segments

Monotonization makes bounding-boxes very useful
Example of monotonized segment

$p + 1 - 1$
Computing intersections

- Split into monotonic segments during preprocess
  - Parts with \( c_m(t_m) = (x_m(t_m), y_m(t_m)) \), \( t_m \in [0, 1] \)
  - \( x'(t_m) \) and \( y'(t_m) \) have no roots for \( t_m \in [0, 1] \)
  - Requires solving linear or quadratic equations

- Simpler intersection test during rendering
  - One intersection at \( t_{mi} \in [0, 1] \) if and only if
    \[
    \min (y_m(0), y_m(1)) < y_s \leq \max (y_m(0), y_m(1))
    \]
  - Find \( t_{mi} \) robustly (e.g., safe Newton–Raphson)
  - Check that \( x(t_{mi}) > x_s \)
  - Test sign of \( y_m(1) - y_m(0) \) to inc/dec winding number
Implicit linear test

- Outside bounding box, trivial
- Inside bounding box, use implicitization

\[
L(x, y) = a_{10}(x - x_0) + a_{01}(y - y_0)
\]

\[
s = \text{sign}(y_1 - y_0)
\]

\[
a_{10} = s(y_1 - y_0)
\]

\[
a_{01} = s(x_0 - x_1)
\]
Implicit linear test

• Outside bounding box, trivial
• Inside bounding box, use implicitization

\[ L(x, y) = a_{10}(x - x_0) + a_{01}(y - y_0) \]
\[ s = \text{sign}(y_1 - y_0) \]
\[ a_{10} = s(y_1 - y_0) \]
\[ a_{01} = s(x_0 - x_1) \]
What about curves?

• Must be careful
  • Parametrization is *local* to [0,1]
  • Implicitization is *global*
Monotonic segment with no inflections

Theorem: Monotonic segments with no inflections cannot cross line connecting endpoints for $t \in (0, 1)$

- After split, 8 configurations
  - Goes up/down
  - Connects diagonal/anti-diagonal
  - Entirely to left/right of diagonal
Monotonic quadratics

Theorem: Quadratic $q(t)$ cannot reenter triangle $Q_0Q_1Q_2$ for $t \not\in [0, 1]$

Theorem: Quadratic $q(t)$ cannot reenter triangle $Q_0PQP_2$ for $t \not\in [0, 1]$

$L(x, y) < 0 \text{ or } Q(x, y) < 0$

$Q(x, y) = (a_{10} + a_{20}x)x + (a_{01} + a_{11}x + a_{02}y)y$
Monotonic quadratics

Theorem: Quadratic \( q(t) \) cannot reenter triangle \( Q_0Q_1Q_2 \) for \( t \not\in [0, 1] \)

Theorem: Quadratic \( q(t) \) cannot reenter triangle \( Q_0PQ_2 \) for \( t \not\in [0, 1] \)

\[
L(x, y) < 0 \text{ and } Q(x, y) < 0
\]

\[
Q(x, y) = (a_{10} + a_{20}x)x + (a_{01} + a_{11}x + a_{02}y)y
\]
Abstract segments

- Similar setup for cubics and rational quadratics
- Primitive of choice for vector graphics pipeline
- Encapsulates monotonic segment $s$
  - Bounding-box, up-down, precomputed implicitization
  - Method $s$.winding$(x,y)$
    - Returns $+1$ or $-1$ if ray from $(x,y)$ to $(\infty,y)$ hits, $0$ otherwise
Sampling algorithm

for all samples \((x,y)\)

for all shapes

\[\text{winding number} = 0\]

for all segments \(s\)

\[\text{winding number} += s.\text{widing}(x,y)\]

if winding number implies inside

blend paint into output
The right acceleration data structure

THE SHORTCUT TREE
Acceleration data structure

[Nehab & Hoppe 2008]: Regular grid

Ours [Ganacim et al. 2014]: Quadtree
Sampling algorithm

for all samples

    find cell containing sample

    for subset of shapes in cell

        winding number = 0

        for subset of segments s in cell

            winding number += s.winding(x,y)

        if winding number implies inside

            blend paint into output
What goes on each cell?

• Specialized subset of illustration
• Everything that is needed to render cell region

Invariant: The winding number of all paths about all samples in the cell region, computed from the cell contents, is exactly the same as in the complete illustration
Clippnig is overkill

We only cast rays to the right
What goes on each cell?

- Specialized subset of illustration
- Everything that is needed to render cell region
- *Only* what is needed to render cell region

Invariant: The winding number of all paths about all samples in the cell region, computed from the cell contents, is exactly the same as in the complete illustration
What about content to right of cell?
Cannot be simply discarded

Case 1

Case 2

Case 3
Could use clipping

[Sutherland & Hodgman 1974]
Shortcut simplification

Case 1
Input contour

Equivalent but non-local

Equivalent and locally decided

Case 2

Winding increment: 0

Winding increment: 0

Winding increment: -1

Case 3

[Nehab & Hoppe 2008] [Ganacim et al. 2014]
Correctness of shortcut simplification

Theorem: Flipping a shortcut segment adds \( \pm 1 \) to all winding numbers in the cell

Corollary: There is an integer \( k \) that restores the invariant

Corollary: The residual between winding number of input and simplification at any point gives \( k \)
Shortcut simplification summary

• Include segment if and only if it overlaps with cell
• Add shortcut *up* for segments that cross border
• Add winding increments for segments that cross border

Shortcut simplification preserves the invariant

A

B
Ours [Ganacim et al. 2014] (preserve original segments)
Subdivision

- Child cells inherit winding increments from parent
- Include those segments that overlap with each child cell
- Check border crossings with \( A \), \( B \), \( C \), \( D \) for shortcuts
- Check border crossings with \( E \), \( F \), \( G \) for increments

Subdivision preserves the invariant
Example subdivision

Winding increment: -1

Shortcut segment
Parallel subdivision

Segment-parallel classification

Parallel-scan followed by segment-parallel copy
Sample sharing

SAMPLE SCHEDULER
Sample sharing

Unit length kernel

Length 4 kernel: loop by pixel

Length 4 kernel: loop by unit sample group
Parallel sample scheduling

Find tree cell for each sample

Group samples by cell

Compute sample colors...

... and integrate
RESULTS
Alias, noise, and gamma

Most renderers
Gamma,
Box weights

NVPR
Linear, 8spp multisampling

[Nehab & Hoppe 2008]
Linear, 1spp,
Prefilter approximation

Ours [Ganacim et al. 2014]
Linear, 32x4x4 spp,
Cardinal Cubic B-spline weights
Conflation

- Resolving each path to pixels before blending causes artifacts
- Correct results require blending each sample independently
  - NVPR also correct
Examples of user-defined warps

- Warp each sample position
- *Not* a post-processing step
- Engages sample scheduler
Rendering time (ms)

- Ours 32x
- NVPR 8x

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Car</th>
<th>Drops</th>
<th>Paper 1</th>
<th>Countour</th>
<th>Paris 30k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low complexity</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>Medium complexity</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>150</td>
</tr>
<tr>
<td>High complexity</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>250</td>
</tr>
</tbody>
</table>
Left out of talk

• New algorithm for rendering with clip-paths
  • Full SVG semantics, no stack or recursion

• Parallel pruning algorithm in preprocessing
  • Eliminates occluded or clipped paths from cells

• Please see paper for other omitted details
Several ideas for future work

• Support for rational cubics (enable object-space warps)
• Support for mesh-based gradients
• Parallel stroke-to-fill conversion
• Transparency groups
• Subpixel rendering (e.g., ClearType)
• Raster effects over groups (e.g., Gaussian Blur)
• Different subdivision strategies (e.g., kd-tree)
• Port back to CPU with multi-threaded vector code
• Hardware implementation
Conclusions

• Fully parallel vector graphics rendering pipeline
• Interactive preprocessing times
• Support for user-defined warps
• Unprecedented output quality
• Best option for complex illustrations
• Source-code available

www.impa.br/~diego/projects/GanEtAl14/