Einführung in Visual Computing

Clipping

Werner Purgathofer
Viewing in the Rendering Pipeline

scene objects in object space

object capture/creation
modeling
viewing
projection

transformed vertices in clip space

clipping + homogenization

scene in normalized device coordinates

viewport transformation
rasterization
shading

vertex stage ("vertex shader")

pixel stage ("fragment shader")
raster image in pixel coordinates
Overview: Clipping

- line clipping
- polygon clipping
- triangle clipping
partly visible or completely invisible parts
must not be ignored and must not be drawn

⇒ must be cut off (as early as possible)
Clipping Operations

- remove objects outside a clip window
  - clip window: rectangle, polygon, curved boundaries
  - applied somewhere in the viewing pipeline
  - can be combined with scan conversion
  - objects to clip: points, lines, triangles, polygons, curves, text, ...

Werner Purkathofer
3 Principle Possibilities for Clipping

- **analytically** in *world* coordinates
  - reduces WC $\rightarrow$ DC transformations

- **analytically** in *clip* coordinates
  - simple comparisons

- *during raster conversation*
  - as part of the rasterization algorithm
  - may be efficient for complex primitives
Line Clipping (1)

before clipping

after clipping

[line clipping against a rectangular clip window]
goals

- eliminate simple cases fast
- avoid intersection calculations

for endpoints \((x_0, y_0), (x_{\text{end}}, y_{\text{end}})\)
intersect parametric representation

\[
    x = x_0 + u \cdot (x_{\text{end}} - x_0) \\
y = y_0 + u \cdot (y_{\text{end}} - y_0)
\]

with window borders:
intersection \(\iff 0 < u < 1\)
Cohen-Sutherland Line Clipping

- assignment of region codes to line vertices

- bit1: left
- bit2: right
- bit3: below
- bit4: above

binary region codes assigned to line endpoints according to relative position with respect to the clipping rectangle
Cohen-Sutherland Line Clipping

- “or” of codes of both points
  \( = 0000 \Rightarrow \text{line entirely visible} \)

- “and” of codes of both points
  \( \neq 0000 \Rightarrow \text{line entirely invisible} \)

- all others \( \Rightarrow \text{intersect!} \)
lines extending from one coordinate region to another may pass through the clip window, or they may intersect clipping boundaries without entering the window.
Cohen-Sutherland Line Clipping

- remaining lines
  - intersection test with bounding lines of clipping window
  - left, right, bottom, top
  - discard an outside part
  - repeat intersection test up to four times

vertical: \( x = x_{\text{wmin}} \)
\[ y = y_0 + m(x_{\text{wmin}} - x_0) \]

vertical: \( y = y_0 + m(x_{\text{wmin}} - x_0) \), \( y = y_0 + m(x_{\text{wmax}} - x_0) \)

horizontal: \( x = x_0 + (y_{\text{wmin}} - y_0)/m \), \( x = x_0 + (y_{\text{wmax}} - y_0)/m \)
Cohen-Sutherland Line Clipping

\[ y = y_0 + m(x_{w\text{min}} - x_0), \quad y = y_0 + m(x_{w\text{max}} - x_0) \]

\[ x = x_0 + (y_{w\text{min}} - y_0)/m, \quad x = x_0 + (y_{w\text{max}} - y_0)/m \]

passes through clipping window

intersects boundaries without entering clipping window
Polygon Clipping

- modification of line clipping
- goal: one or more closed areas

display of a polygon processed by a line-clipping algorithm

display of a correctly clipped polygon
Sutherland-Hodgman Polygon Clipping

- processing polygon boundary as a whole against each window edge
- output: list of vertices

clipping a polygon against successive window boundaries
Sutherland-Hodgman Polygon Clipping

- four possible edge cases

- successive processing of pairs of polygon vertices against the left window boundary

Output:
1. Out → In: $V_{\text{new}}, V_2$
2. In → In: $V_2$
3. In → Out: $V_{\text{new}}$
4. Out → Out: No output
Sutherland-Hodgman Polygon Clipping

for 1 edge:

1. \( V_2 := 1^{st} \) vertex
2. \( V_1 := V_2 \)
3. \( V_2 := \) next vertex
4. \( V_2 \) visible?
   - no
5. \( V_1 \) visible?
   - no
Sutherland-Hodgman Polygon Clipping

for 1 edge:

\[ V_2 := 1^{\text{st}} \text{ vertex} \]

\[ V_1 := V_2 \]

\[ V_2 := \text{next vertex} \]

\[ V_2 \text{ visible?} \]

no

\[ V_1 \text{ visible?} \]

no

\[ V_{\text{new}} = \text{clip edge} \cap V_1 V_2 \rightarrow \text{result list} \]

yes
Sutherland-Hodgman Polygon Clipping

for 1 edge:

- $V_2 := 1^{st}$ vertex
- $V_1 := V_2$
- $V_2 :=$ next vertex
- $V_2$ visible? no
- $V_1$ visible? yes

- $V'_1 = \text{clip edge} \cap V_1V_2 \rightarrow \text{result list}$
- $V_2 \rightarrow \text{result list}$
- $V_1$ visible? yes
- $V_1$ visible? yes
for 1 edge:

\[
\begin{align*}
V_2 &:= 1^{st} \text{ vertex} \\
V_1 &:= V_2 \\
V_2 &:= \text{next vertex} \\
V_2 \text{ visible?} &\quad \text{no} \\
V_1 \text{ visible?} &\quad \text{no} \\
V'_{1} &:= \text{clip edge} \cap V_1V_2 \\
&\quad \rightarrow \text{result list}
\end{align*}
\]
clipping a polygon against the left boundary of a window, starting with vertex 1.

primed numbers are used to label the points in the output vertex list for this window boundary.
the polygon is clipped against each of the 4 borders separately, that would produce 3 intermediate results.

by calling the 4 tests *recursively*, (or by using a clipping pipeline) every result point is immediately processed on, so that only *one* result list is produced
pipeline of boundary clippers to avoid intermediate vertex lists

Processing the polygon vertices through a boundary-clipping pipeline. After all vertices are processed through the pipeline, the vertex list for the clipped polygon is \{1', 2, 2', 2''\}
Sutherland-Hodgman Polygon Clipping

- extraneous lines for concave polygons:
  - split into separate parts  or
  - final check of output vertex list

clipping the concave polygon with the Sutherland-Hodgeman clipper produces three connected areas
Clipping of Triangles

- often b-reps are “triangle soups”
- clipping a triangle → triangle(s)
- 4 possible cases:
  - inside
  - outside
  - triangle
  - quadrilateral → 2 triangles
corner cases need no extra handling!
From Object Space to Screen Space

- **modeling transformation**
- **world space**
- **clip space**
- **camera transformation**
- **camera space**
- **projection transformation**
- **viewport transformation**
- **screen space**

„view frustum“
Clipping in Clip-Space

- clipping against \( x = \pm 1, \ y = \pm 1, \ z = \pm 1 \)
- \((x,y,z)\) inside?
- \(\implies\) only compare one value per border!

- is done \textit{before} homogenization:
  - \( x = \pm h, \ y = \pm h, \ z = \pm h \)
  - clips points that are behind the camera!
  - reduces homogenization divisions
Einführung in Visual Computing

186.822

Antialiasing

Werner Purgathofer
Antialiasing in the Rendering Pipeline

scene objects in object space

transformed vertices in clip space

scene in normalized device coordinates

raster image in pixel coordinates

object capture/creation

modeling

viewing

projection

clipping + homogenization

viewport transformation

rasterization

shading

vertex stage ("vertex shader")

pixel stage ("fragment shader")
what is aliasing? ['eiliæsinə]  
what is the reason for aliasing?  
what can we do against it?
What is Aliasing?

errors that are caused by the discretization of analog data to digital data

- **too bad resolution**
- **too few colors**
- **too few images/sec**
- **geometric errors**
- **numeric errors**
Aliasing: Staircase Effect
Various Aliasing Effects
Aliasing from too few Colors

artificial color borders can appear
Aliasing in Animations

- jumping images
- "worming"
- backwards rotating wheels
1. improve the devices
   - higher resolution
   - more color levels
   - faster image sequence

2. improve the images = antialiasing
   - postprocessing
   - prefiltering

Expensive or incompatible

Software!
a signal can only be reconstructed without information loss if the **sampling frequency** is at least twice the highest frequency of the signal

this border frequency is called "**Nyquist Limit**"
Nyquist-Shannon Sampling Theorem

original signal

reconstructed signal

sampling rate $\Delta$

Nyquist sampling interval

$\Delta$
Antialiasing: Nyquist Sampling Frequency

A signal can only be reconstructed without information loss if the **sampling frequency** is at least twice the highest frequency of the signal.

Nyquist sampling frequency: \( f_s = 2 f_{\text{max}} \)

\[
\Delta x_s = \frac{\Delta x_{\text{cycle}}}{2}
\]

with

\[
\Delta x_{\text{cycle}} = \frac{1}{f_{\text{max}}}
\]

i.e. sampling interval \( \leq \) one-half cycle interval
Antialiasing Strategies

- supersampling straight-line segments
- subpixel weighting masks
- area sampling straight-line segments
- filtering techniques
- compensating for line-intensity differences
- antialiasing area boundaries
  - (adjusting boundary pixel positions)
  - adjusting boundary pixel intensity
Antialiasing: Supersampling Lines

**mathematical line**

3 = max. intensity

... = min. intensity

**line of finite width**

9 = max. intensity

... = min. intensity
Antialiasing
- calculate the pixel coverage exactly
- can be done with incremental schemes

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>0%</th>
<th>43%</th>
</tr>
</thead>
<tbody>
<tr>
<td>15%</td>
<td>71%</td>
<td>84%</td>
<td></td>
</tr>
<tr>
<td>90%</td>
<td>52%</td>
<td>3%</td>
<td></td>
</tr>
</tbody>
</table>
Antialiasing: Pixel Weighting Masks

- more weight for center subpixels
- must be divided by sum of weights
- subpixel grids can also include some neighboring pixels

Relative weights for a grid of 3x3 subpixels

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Antialiasing: Filtering Techniques

continuous overlapping weighting functions to calculate the antialiased values with integrals

box filter       cone filter    Gaussian filter
Antialiasing: Intensity Differences

- unequal line lengths displayed with the same number of pixels in each line/row have different intensities
- proper antialiasing compensates for that!
Antialiasing Area Boundaries
Antialiasing Area Boundaries (1)

adjacent pixel intensities along an area boundary

*alternative 1:*
supersampling

Scan Line 1
Scan Line 2

Subdivided Pixel Area
**alternative 2:** similar to Bresenham algorithm

\[ p' = y - y_{\text{mid}} = [m(x_k + 1) + b] - (y_k + 0.5) \]

\[ p' < 0 \Rightarrow y \text{ closer to } y_k \]
\[ p' > 0 \Rightarrow y \text{ closer to } y_{k+1} \]

\[ p = p' + (1-m) \]
\[ p < l-m \Rightarrow y \text{ closer to } y_k \]
\[ p > l-m \Rightarrow y \text{ closer to } y_{k+1} \]

( and \( p \in [0,1] \))
\[ p = p' + (1 - m) = [m(x_k + 1) + b] - (y_k + 0.5) + (1 - m) = mx_k + b - y_k + 0.5 = mx_k + b - (y_k - 0.5) \]

Antialiasing Area Boundaries (3)
Antialiasing Area Boundaries (4)
Antialiasing Examples

- aliased
- antialiasing

- aliased
- antialiasing

- aliased
- antialiasing
Antialiasing Examples