



# **SIGGRAPH2006**

## A Brief Introduction to OpenVG

Randi Rost, Intel

Daniel Rice, Sun Microsystems, Inc.

# OpenVG Timeline

- OpenVG 1.0 was ratified in August, 2005
- OpenVG 1.0.1 will be ratified in the next 2 months
  - This specification contains clarifications only
- OpenVG 1.0.1 Conformance Tests will be available in August
- OpenVG 1.1 is being defined now
- Possible 1.1 Features Include:
  - Accelerated text
  - Flash-compatible rendering
  - Willing to consider other new features
- Target date for 1.1 is Q1, 2007

# OpenVG Overview

- **Provides a 2D vector drawing model**
  - Similar to SVG, PostScript, PDF, Java2D, Flash
- **Target applications:**
  - SVG viewer, Portable mapping, E-book reader, Games, UI, etc.
- **Similarities to OpenGL ES (1.1)**
  - Similar API style
  - State machine
  - Well-defined rendering pipeline
  - Uses EGL for access to rendering context and draw buffer
- **Differences from OpenGL**
  - Focus is on 2D vector graphics, not 3D

# The OpenVG Pipeline

- OpenVG defines a hardware pipeline for paths and images
- Path Definition & Setting of API Parameters
- Stroking
  - Line width, joins & caps, dashing, etc.
- Transformation
  - 2x3 and 3x3 transformations
- Rasterization
- Clipping & Masking
  - Scissoring rectangles, alpha mask
- Paint Generation
  - Flat color, gradient, or pattern paint

Path Definition &  
Setting of API Parameters

Stroking

Transformation

Rasterization

Clipping and Masking

Paint Generation

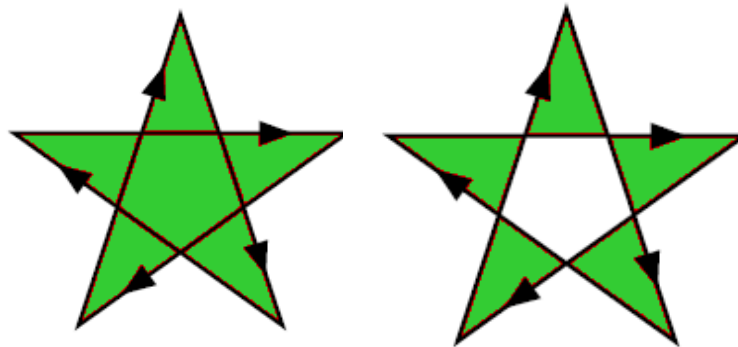
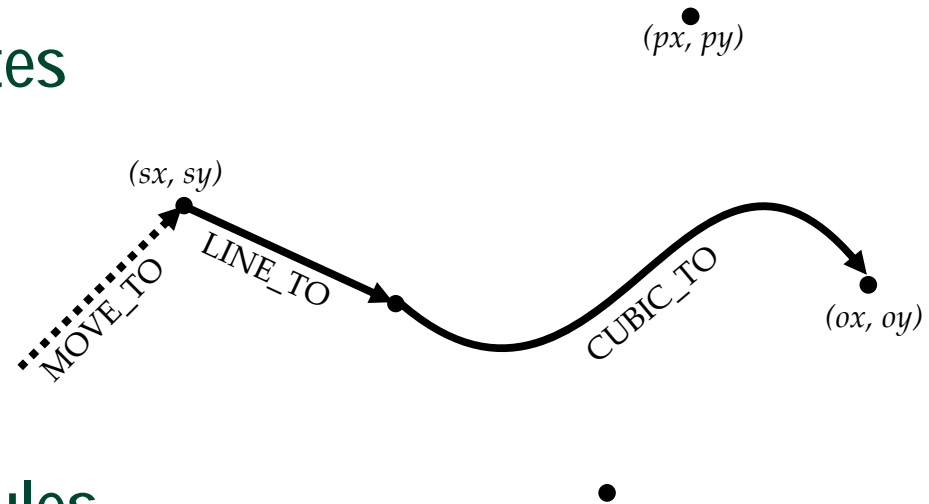
Blending

Dithering



# Path Definition

- MOVE\_TO, LINE\_TO, QUAD\_TO, CUBIC\_TO, CLOSE\_PATH
- Elliptical Arcs
- Absolute / Relative Coordinates
- Path Queries:
  - Bounding Boxes
  - Transformed Bounding Boxes
  - Point along path
  - Tangent along path
- Non-Zero and Even-Odd fill rules



# Setting API Parameters

- OpenVG follows the OpenGL model:

- `vg{Get,Set}{f,i,fv,iv}`
- `vg{Get,Set}Parameter{f,i,fv,iv}`

- Settable parameters:

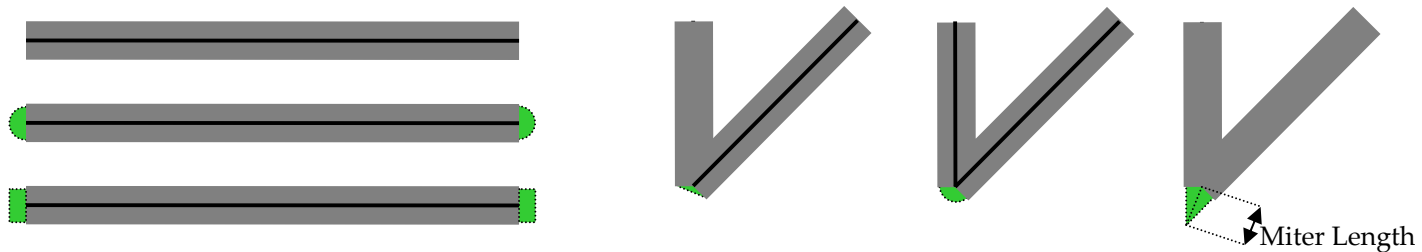
- `VG_MATRIX_MODE, VG_FILL_RULE, VG_IMAGE_QUALITY, VG_RENDERING_QUALITY, VG_BLEND_MODE, VG_IMAGE_MODE, VG_SCISSOR_RECTS, VG_STROKE_LINE_WIDTH, VG_STROKE_CAP_STYLE, VG_STROKE_JOIN_STYLE, VG_STROKE_MITER_LIMIT, VG_STROKE_DASH_PATTERN, VG_STROKE_DASH_PHASE, VG_TILE_FILL_COLOR, VG_CLEAR_COLOR, VG_MASKING, VG_SCISSORING, VG_PIXEL_LAYOUT, VG_FILTER_FORMAT_LINEAR, VG_FILTER_FORMAT_PREMULTIPLIED, VG_FILTER_CHANNEL_MASK`

- Read-only values:

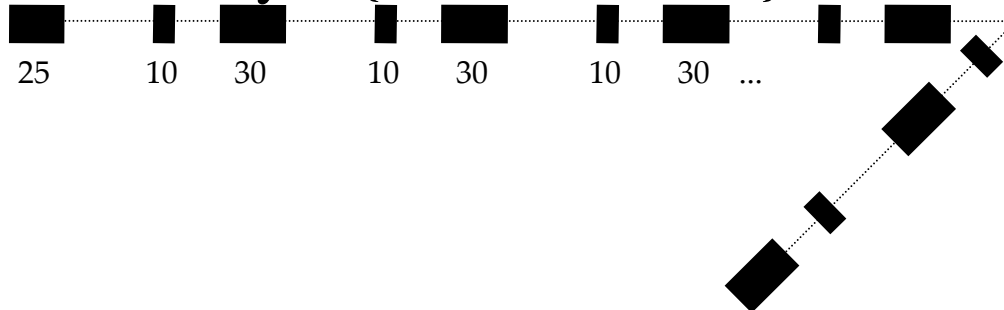
- `VG_MAX_SCISSOR_RECTS, VG_MAX_DASH_COUNT, VG_MAX_KERNEL_SIZE, VG_MAX_SEPARABLE_KERNEL_SIZE, VG_MAX_COLOR_RAMP_STOPS, VG_MAX_IMAGE_WIDTH, VG_MAX_IMAGE_HEIGHT, VG_MAX_IMAGE_PIXELS, VG_MAX_IMAGE_BYTES, VG_MAX_FLOAT`

# Stroking

- Stroking takes a path and defines an outline around it:
  - Line Width
  - End cap style (Butt, Round, or Square)
  - Line join style (Bevel, Round, or Miter)
  - Miter limit (to convert long miters to bevels)
  - Dash array and offset



Dash array = { 10, 20, 30, 40 } / Dash Phase = 35



# Transformations

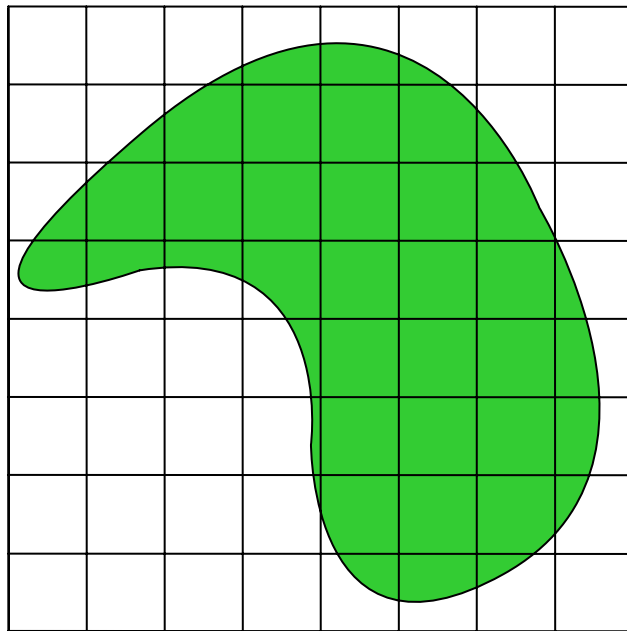
- Paths use 2x3 affine transformations
- Images use 3x3 perspective transformations
- Transformation functions are similar to OpenGL:
  - `vgLoadIdentity`
  - `vgLoadMatrix`
  - `vgGetMatrix`
  - `vgMultMatrix`
  - `vgScale`
  - `vgRotate`
  - `vgTranslate`
  - `vgShear`



$$\begin{bmatrix} 1.080 & 0.101 & 0 \\ 0.209 & 0.691 & 0 \\ 1.28 \times 10^{-3} & -1.19 \times 10^{-3} & 1 \end{bmatrix}$$

# Rasterization (continued)

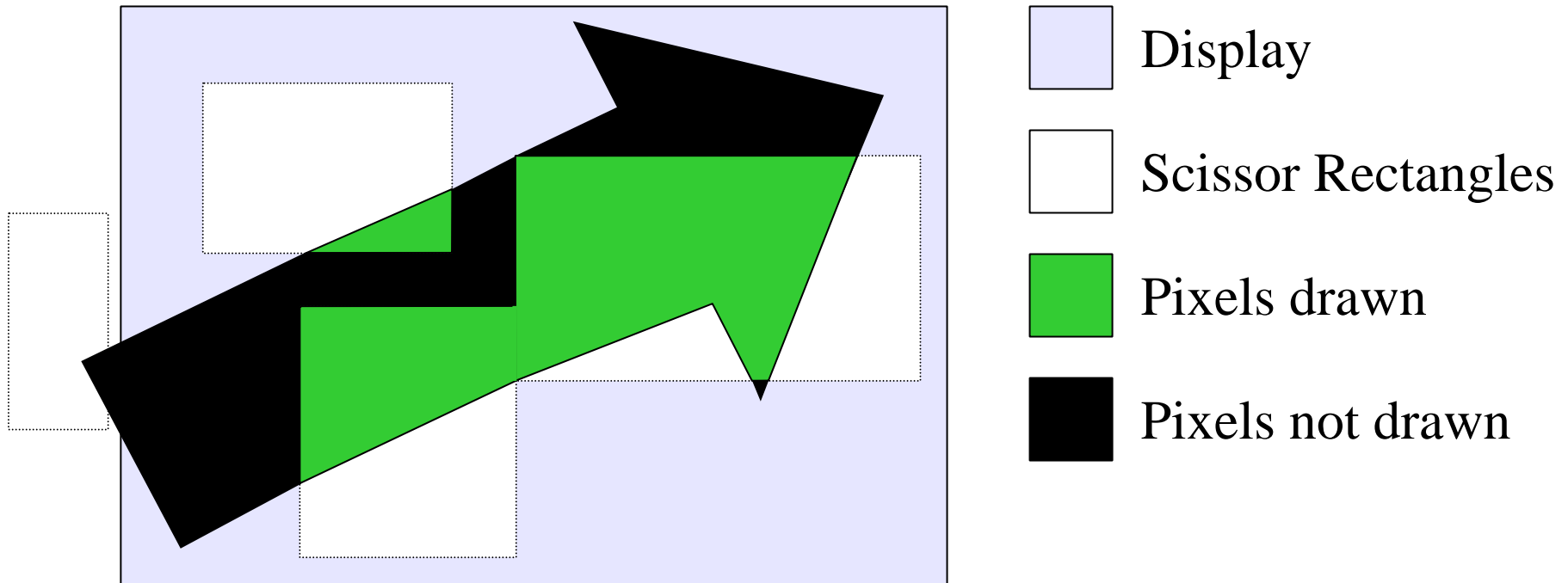
- The goal of rasterization is to determine a filtered alpha value for each pixel, based on the geometry around that pixel
- Filters may be up to 3 pixels in diameter



0	0	.1	.4	.5	.2	0	0
0	.3	.8	1	1	.9	.4	0
.4	.8	1	1	1	1	.8	0
.6	.4	.3	.7	1	1	1	.3
0	0	0	.2	1	1	1	.5
0	0	0	.1	1	1	1	.5
0	0	0	.1	.9	1	.9	.2
0	0	0	0	.3	.5	.2	0

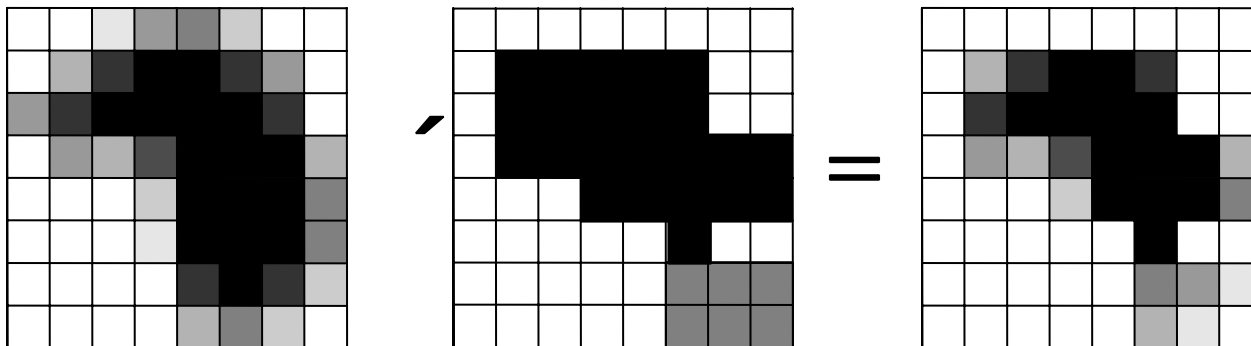
# Scissoring

- Only pixels inside a set of scissor rectangles are drawn
- Scissoring is disabled by default



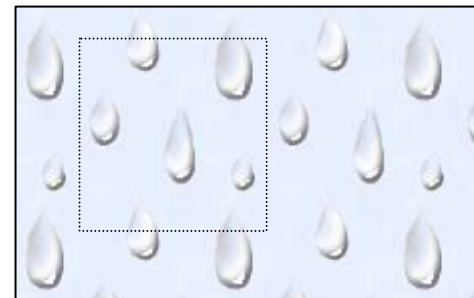
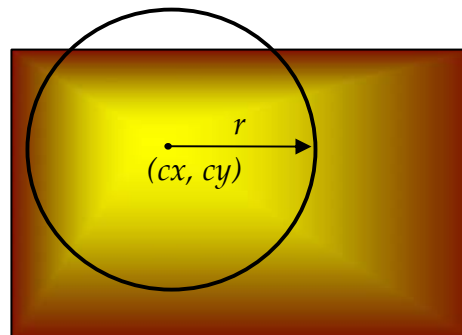
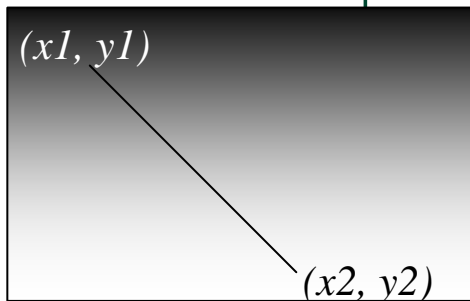
# Masking

- In addition to scissoring, a per-pixel mask may be applied
- The mask has an alpha value at each pixel that is multiplied by the alpha from the rendering stage
- May be used to “cut out” an area, create area transitions
- Mask values may be modified using image data
  - Fill, Clear, Set, Add, Subtract, Intersect



# Paint Generation

- Paint is generated pixel-by-pixel and applied to geometry
- The alpha from the previous stage (rendering + masking) is used to determine how much paint to apply
- Separate paint objects for stroking, filling
- Paint is transformed by an affine transform
- Four types of paint are supported:
  - Flat color paint
  - Linear gradient paint: points  $(x_1, y_1)$  and  $(x_2, y_2)$ , color ramp
  - Radial gradient paint: center  $(x, y)$ , focus  $(x, y)$ , radius, color ramp
  - Pattern paint based on an image, tiling mode

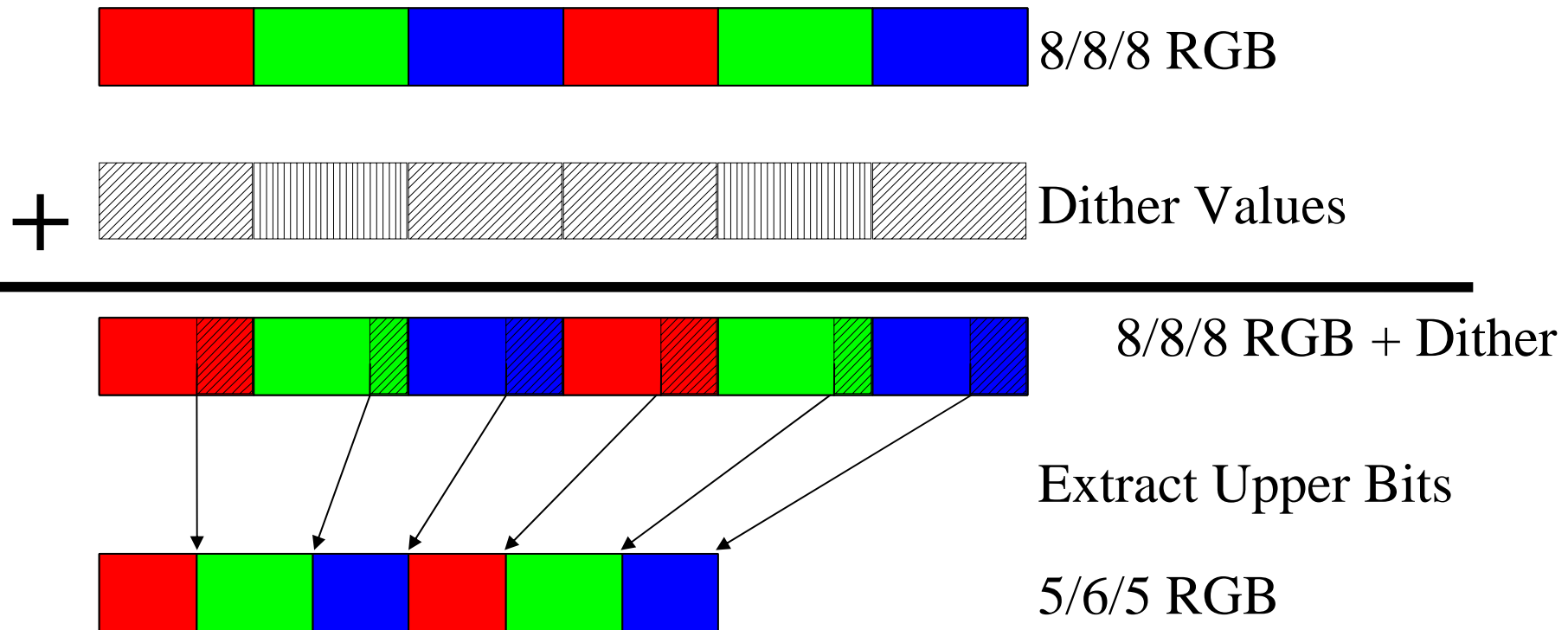


# Blending

- Combine masked alpha from path with paint alpha
- Blend the result onto the drawing surface
- Blending is a function of:
  - The paint (R, G, B) color
  - The masked alpha value (path alpha  $\times$  mask alpha  $\times$  paint alpha)
  - The destination (R, G, B) color
  - The destination alpha value (1 if no stored alpha)
- There are 8 blending functions:
  - Porter-Duff "source" mode (copy source to destination)
  - Porter-Duff "source **over** destination"/ "destination **over** source"
  - Porter-Duff "source **in** destination"/ "destination **in** source"
  - Lighten, Darken, Multiply, Screen
  - Additive (add pixel values, add alpha up to 1)

# Dithering

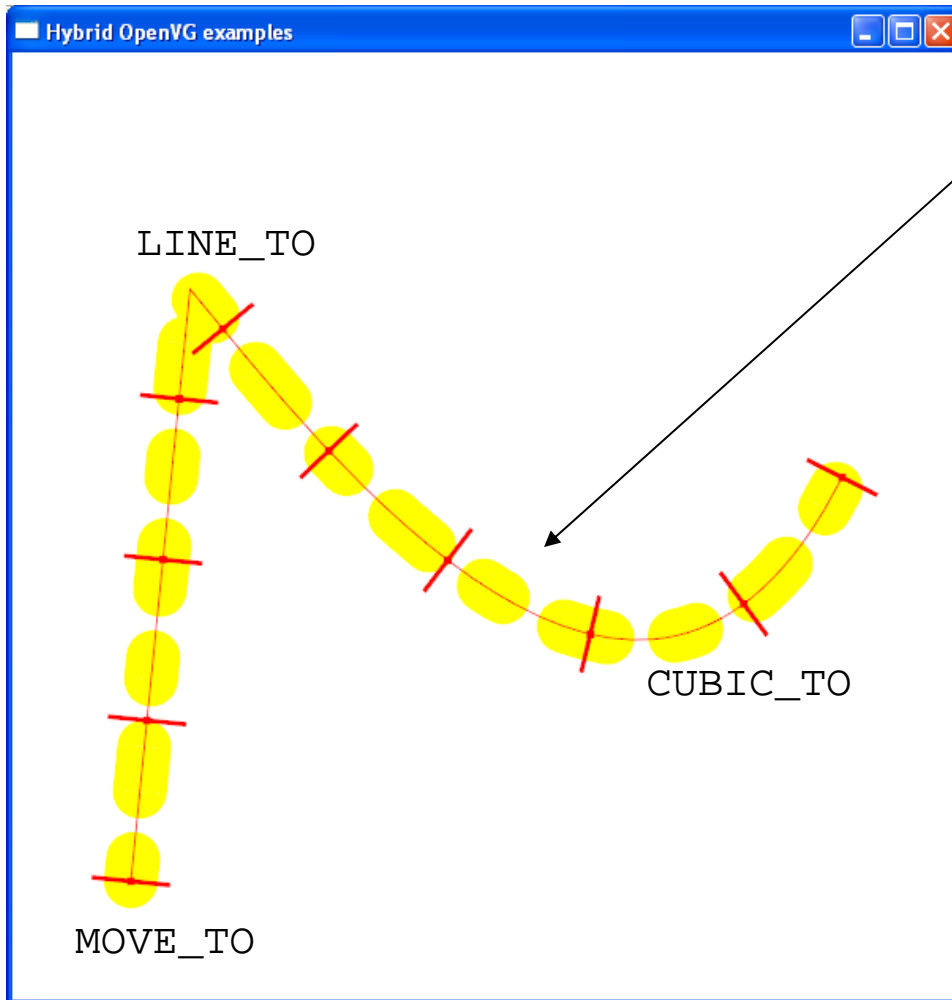
- As a final stage, the bit depth of pixels may be reduced using dithering
- The details of dithering are platform-specific



# Images

- **Images are defined using one of 13 pixel formats**
  - Linear or non-linear (sRGB) color spaces
  - Linear or non-linear grayscale
  - Pre-multiplied or non-premultiplied alpha
  - 8/8/8, 5/6/5, 5/5/5/1, 4/4/4/4 bit depths (< 8 non-linear color only)
  - 1-bit Black & White (e.g., for Fax applications)
- **Images may be stored in accelerated memory**
- **Image filters may be applied:**
  - Color Matrix
  - Convolve, Separable Convolve, Gaussian Blur
  - Lookup, LookupSingle
- **Images may be drawn in perspective**
- **Image may be used as a stencil to apply paint**
  - Very useful for drawing anti-aliased text
- **Image and paint colors may be multiplied together**

# Stroking



```
VGfloat d[] = { 5, 15, 10, 15 };  
vgSetfv(VG_DASH_PATTERN, 4, d);
```

```
VGPath path = vgCreatePath(...);  
cmd[0] = VG_MOVE_TO_ABS;  
cmd[1] = VG_LINE_TO_ABS;  
cmd[2] = VG_CUBIC_TO_ABS;  
coord[0] = ...;  
vgAppendPathData(...)  
vgDrawPath(path, VG_STROKE_PATH);
```

# Creating a Path

```
VGubyte * commands;
VGfloat * coords;
VGint numCmds, numCoords;

// 0,0 is O.K. for numCommands, numCoords
VGPath path = vgCreatePath(VG_PATH_FORMAT_STANDARD,
                           VG_PATH_DATATYPE_F,
                           1.0f, 0.0f, // scale,bias
                           numCmds, numCoords,
                           VG_PATH_CAPABILITY_ALL);

commands[0] = VG_MOVE_TO_ABS;
coords[0] = ...; coords[1] = ...; /* x,y */
/* ... */

vgAppendPathData(path, numCmds, commands, coords);
```

# Creating Color Paint

```
VGfloat color[] = { 1.0f, 1.0f, 0.0f, 1.0f }; /* RGBA */
VGPaint colorPaint = vgCreatePaint();

/* Paint Type */
vgSetParameteri(paint,
                VG_PAINT_TYPE, VG_PAINT_TYPE_COLOR);

/* Paint Color */
vgSetParameterfv(paint, VG_PAINT_COLOR, 4, color);
```

# Setting Stroking Parameters

```
VGfloat lineWidth, miterLimit;  
VGint capStyle, joinStyle;  
VGfloat dashPattern[NUM_DASHES], dashPhase;  
  
vgSetParameterf(VG_STROKE_LINE_WIDTH, lineWidth);  
vgSetParameteri(VG_STROKE_CAP_STYLE, capStyle);  
vgSetParameteri(VG_STROKE_JOIN_STYLE, joinStyle);  
vgSetParameterf(VG_STROKE_MITER_LIMIT, miterLimit);  
vgSetParameterfv(VG_STROKE_DASH_PATTERN,  
                 NUM_DASHES, dashPattern);  
vgSetParameterfv(VG_STROKE_DASH_PATTERN,  
                 0, (VGfloat *) 0);  
vgSetParameterf(VG_STROKE_DASH_PHASE, dashPhase);
```

# Drawing the Path

```
VGPath path;
VGPaint fillPaint, strokePaint;
VGboolean doFill, doStroke;

if (doFill) {
    vgSetPaint(fillPaint, VG_FILL_PATH);
}
if (doStroke) {
    vgSetPaint(strokePaint, VG_STROKE_PATH);
}
if (doFill || doStroke) {
    vgDrawPath(path, (doFill ? VG_FILL_PATH : 0) |
                (doStroke ? VG_STROKE_PATH : 0));
}
```

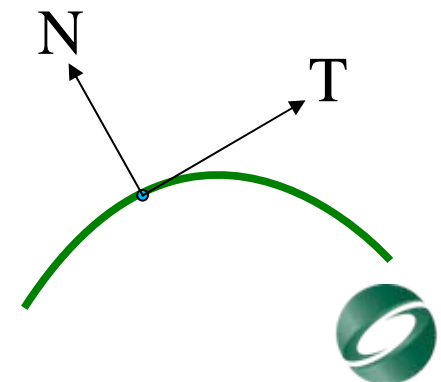
# Finding Points Along the Path

```
/* Determine # of path segments and path length */
VGint numSegments = vgGetParameteri(path,
                                     VG_PATH_NUM_SEGMENTS);
VGfloat length = vgPathLength(path, 0, numSegments);

/* Get equally-spaced points and tangents */
for (i = 0; i < numTicks; i++) {
    VGfloat x, y, tx, ty;
    vgPointAlongPath(path, 0, numSegments,
                    i*length/numTicks,
                    &x, &y, &tx, &ty);
}
```

**Tangent:** draw line from  $(x, y)$  to  $(x + tx, y + ty)$

**Normal:** draw line from  $(x, y)$  to  $(x + ty, y - tx)$



## Further Information

- Read the specification!
- Hybrid has a software implementation you can play with (<http://www.hybrid.fi>)
- <http://www.khronos.org>
- Khronos Tech Talks on OpenVG
  - Wednesday, 2:15-3:30, Room 206A



**SIGGRAPH2006**

The End