Image processing

- PHP Professional Training
- 23th of November 2007

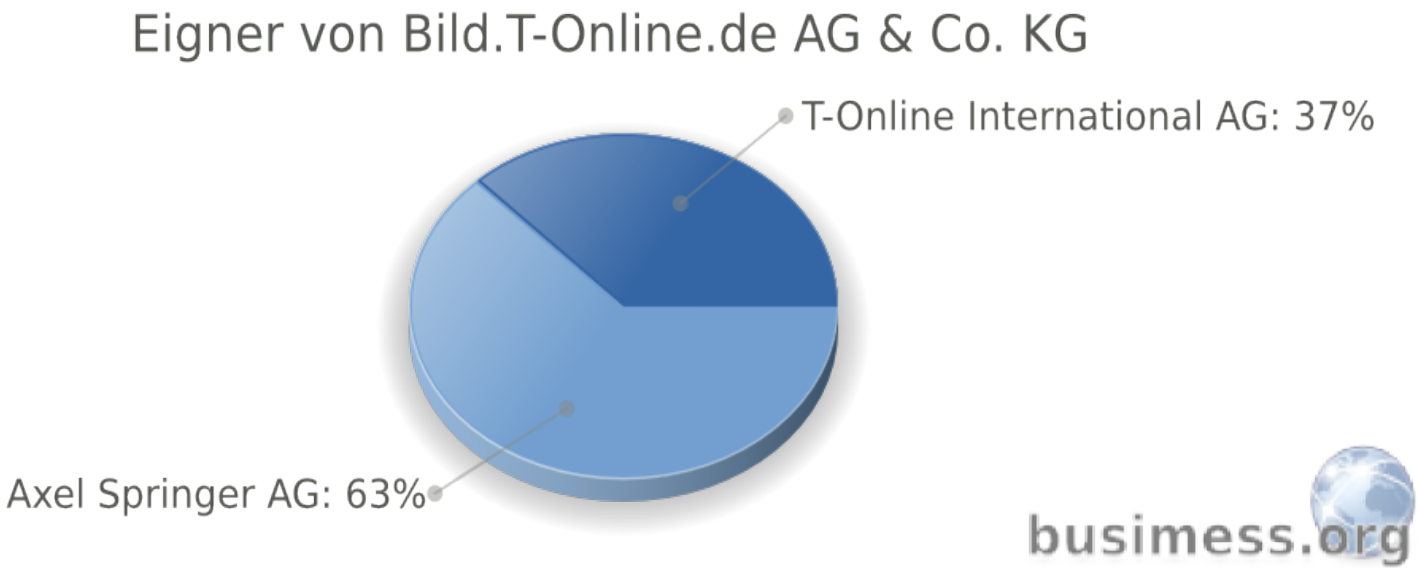
About me

- Kore Nordmann
  - Studying computer science at the University Dortmund
  - Working for eZ systems on eZ components
  - Maintainer and / or Developer in multiple open source projects: Image_3D, KaForkl, ezcGraph, Torii, Busimess, PHPUnit, WCV, ...
    - Image related: Image_3D, KaForkl, ezcGraph

- Goal
  - Write a wrapper to different image formats and libraries with common methods

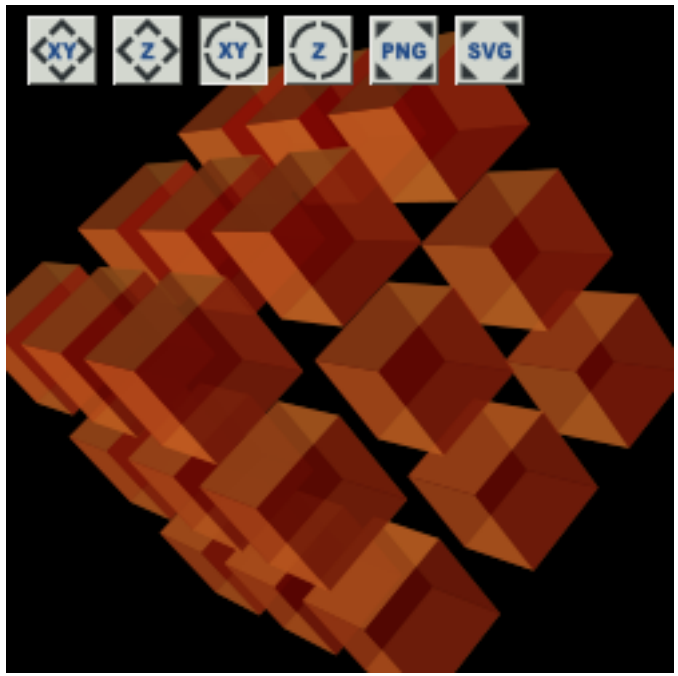
•Usage in ezcGraph

- Supported backends: Flash, GD, SVG



•Usage in Image_3D

- Supported backends: GD, SVG, SVG_Control, Canvas, ASCII



http://westhoffswelt.de/projects/dynamic_canvas_demo.html

Terms

- Image
  - I use it as a generalization of pictures and graphics.
- Picture
  - Images with natural contents, like photos or drawings. Usually there are no or only few clear borders in those images.
- Graphic
  - Computer generated graphics with technical illustrations or charts. They often may contain clear borders.

Agenda:

- Formats
- Libraries
- Basic datastructures
- Creating the surface
- First shape: polygon
- Gradients

Formats

- Vector formats
  - SVG
  - Flash
  - PDF / PS / ...
- Bitmaps
  - GIF
  - JPEG
  - PNG
  - BMP / TIFF / ...

SVG (1)

- Scalable Vector Graphics
  - XML based W3C Standard
    - Large files
    - GZIP compressing
- Easy to generate / user readable
- May include other namespaces
- Scripting using ECMAScript

SVG (2)

- Subset of SVG: TinySVG
  - Common subset known by most clients / devices
- Text rendering is (normally) up to the client
  - No absolute text width / size estimation possible.

SVG (3)

- Pro
  - Open standard
  - Common syntax (XML)
  - Lots of clients
- Cons
  - Slight differencies between clients
  - Font issues
- Common usage
  - Long term usable vector graphics (in the web)

Flash (1)

- Closed standard by Adobe
  - Mainly for animated and interactive web graphics
  - Can be considered as a plain vector format in out case
- Only one real closed source client
  - No support for several platforms
  - Security and privacy issues

Flash (2)

- Pro
  - Rich format installed on a lot of system (98%)
- Con
  - Closed propretary format
  - Clients are not available or easy to develop for all platforms.
  - Bad accessibility
- Common usage
  - Online marketing presentations with mainstream centric target group

PDF / PS / ...

- Also support for vector graphics
- Limited support of graphics features
- Completely different common target group

GIF (1)

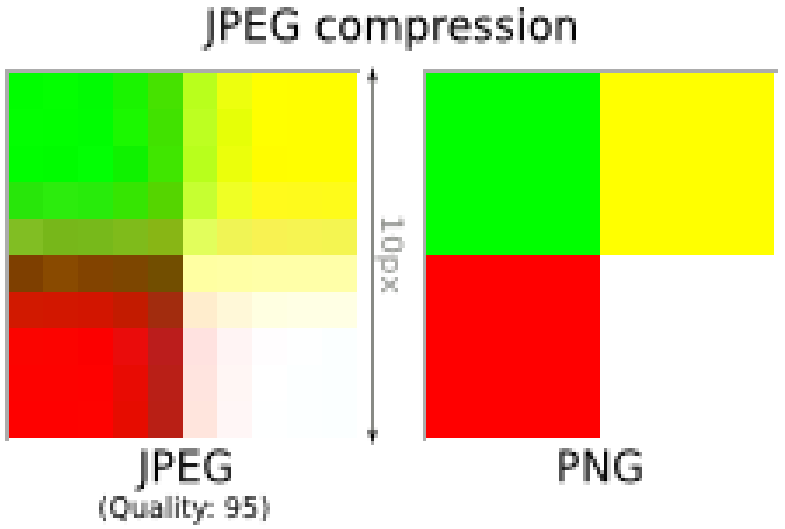
- Patent issues from time to time
- Limited color palette (256 colors)
- No support for semitransparent colors

GIF (2)

- Pro
  - Well known established image format
- Cons
  - Bad transparency support
  - Limited colorspace
- Common usage
  - Web applications, where the author did not know about PNG

JPEG (1)

- (Resolved) Patent issues
- Fourier-Tranformation based compression algorithm
  - Makes it less usable for images with hard edges
- No transparency support



JPEG (2)

- Pros
  - Good compression of pictures
- Cons
  - Bad compresssion and artifacts for graphics.
  - No transparency
- Common usage
  - Pictures

PNG (1)

- Developed and standardized by W3C to replace GIF
  - Ultimate format for graphics in the web
- Lossless compression
- Full RGB colorspace
- 128 alpha channels

PNG (2)

- Pro
  - Well known and established image format
  - Good compression of graphics
  - Full RGB with 128 alpha channels
- Cons
  - Limited support in one browser
- Common usage
  - Graphics in the web.

BMP / TIFF / ...

- Uncompressed, useless, raw formats
  - From a webdeveloper point of view

Agenda:

- Formats
- Libraries
- Basic datastructures
- Creating the surface
- First shape: polygon
- Gradients

Libraries

- ext/gd
- pecl/cairowrapper
- ext/ming
- ext/dom

ext/gd (1)

- Best known extension for image creation in PHP
  - Well documented
  - Lots of examples available
  - Installed nearly everywhere
- Bitmaps are rendered by the library
  - Image quality completely depends on it

ext/gd (2)

- Low quality rendering
  - No anti-aliasing (in most cases)
  - Broken transparency for ellipses (-sectors)
- No gradient support
- Nearly no anti aliasing
- It's damn slow

pecl/cairowrapper (1)

- Renders 2D vectorgraphics
- Used by
  - GTK (since 2.8.0)
  - Firefox (for SVG in 2.*, completely in 3.*)
- Multiple output formats
  - SVG, PDF, PNG, JPEG
  - XWindow, Win32, Glitz (OpenGL), Quartz

pecl/cairowrapper (2)

- Install
  - `pear install pecl/cairo_wrapper-beta`
  - http://pecl.php.net/package/cairo_wrapper
- Supports everything we want.
  - Full transpareny support
  - Excellent anti aliasing
- Really fast rendering

ext/ming

- Creates flash SWF files
- Alpha state since years
- Support for a subset of flash
  - Not matching any specific flash version
- Wrong or missing documentation is usual

ext/DOM

- Implements a well known API for XML handling
- Enables you to modify any part of an document
  - xmlwriter may be faster, but harder to handle (forward only approach)

Agenda:

- Formats
- Libraries
- Basic datastructures
- Creating the surface
- First shape: polygon
- Gradients

Colors (1)

- Wrap the color definitions
  - Create from HEX or array definitions
- Specify the usage of transparency
  - Library dependent: Transparency vs. opacity
  - We define: 255 = full transparency

Colors (2)

```
namespace kn::Graphic;

class Color
{
    public $red;
    public $green;
    public $blue;
    public $alpha;

    public function __construct( $hexString )
    {
        $this->fromHex( $hexString );
    }

    public function __toString()
    {
        return sprintf( '#%02x%02x%02x%02x',
            $this->red,
            $this->green,
            $this->blue,
            $this->alpha
        );
    }
}
```

Colors (3)

- Read hex values

```
public function fromHex( $hexString )
{
    if ( $hexString[0] === '#' )
    {
        $hexString = substr( $hexString, 1 );
    }

    $keys = array( 'red', 'green', 'blue', 'alpha' );
    foreach ( str_split( $hexString, 2) as $nr => $hexValue )
    {
        if ( isset( $keys[$nr] ) )
        {
            $key = $keys[$nr];
            $this->$key = hexdec( $hexValue ) % 256;
        }
    }

    for ( ++$nr; $nr < count( $keys ); ++$nr )
    {
        $key = $keys[$nr];
        $this->$key = 0;
    }
}
```

Coordinates

- Remember the vector class? :)
- ```
  namespace kn::Graphic;

  class Coordinate
  {
      public $x;
      public $y;

      public function __construct( $x, $y )
      {
          $this->x = $x;
          $this->y = $y;
      }
  }
  ```
  ```

The base class (1)

```
namespace kn::Graphic;

require_once 'color.php';
require_once 'coordinate.php';

abstract class Base
{
    public function __construct( $width, $height )
    {
        $this->width  = (int) $width;
        $this->height = (int) $height;

        // Graphic format dependant intialisation
        $this->initialize();
    }

    abstract protected function initialize();

    abstract public function save( $file );
}
```

The base class (2)

- No display() method?
  - Directly displaying of images encourages users not to cache images
  - Image creation always takes a “long” time
    - In the context of web applications

Formats

- Libraries
- Basic datastructures
- Creating the surface
- First shape: polygon
- Gradients

Surface

- Often used synonym for canvas in this context
  - Drawing area
- Implement the methods initialize() and save() for all backends

DOM & SVG - Initialization

```
    protected function initialize()
    {
        $this->dom = new ::DOMDocument( '1.0' );

        $svg = $this->dom-
>createElementNS( 'http://www.w3.org/2000/svg', 'svg' );
        $svg->setAttribute( 'width', '100%' );
        $svg->setAttribute( 'height', '100%' );
        $svg->setAttribute( 'viewbox', "0 0 {$this->width} {$this-
>height}" );
        $svg->setAttribute( 'version', '1.0' );
        $this->dom->appendChild( $svg );

        $this->defs = $this->dom->createElement( 'defs' );
        $this->defs = $svg->appendChild( $this->defs );

        $this->elements = $this->dom->createElement( 'g' );
        $this->elements->setAttribute( 'id', $this->elementPrefix .
'main' );
        $this->elements = $svg->appendChild( $this->elements );
    }
```

DOM & SVG – Save

```
public function save( $file )
{
    $this->dom->save( $file );
}
```

DOM & SVG – Create an empty image

- ```
  $graphic = new Svg( 100, 100 );
  $graphic->save( 'images/example_svg_01.svg' );
  ```
  ```

GD & PNG – Initialize

```
    protected function initialize()
    {
        // Create bigger image respecting the supersampling factor
        $this->image = ::imagecreatetruecolor(
            $this->supersample( $this->width ),
            $this->supersample( $this->height )
        );

        $bgColor = ::imagecolorallocatealpha( $this->image, 255, 255, 255,
127 );
        ::imagealphablending( $this->image, true );
        ::imagesavealpha( $this->image, true );
        ::imagefill( $this->image, 1, 1, $bgColor );

        // Set line thickness to supersampling factor
        ::imagesetthickness(
            $this->image,
            $this->supersampling
        );
    }

    protected function supersample( $value )
    {
        $mod = (int) floor( $this->supersampling / 2 );
        return $value * $this->supersampling - $mod;
    }
```

GD & PNG – Save (1)

```
    public function save( $file )
    {
        if ( $this->supersampling === 1 )
        {
            $destination = $this->image;
        }
        else
        {
            $destination = ::imagecreatetruecolor( $this->width, $this->height );

            // Default to a transparent white background for destination image
            $bgColor = ::imagecolorallocatealpha( $destination, 255, 255, 255,
127 );
            ::imagealphablending( $destination, true );
            ::imagesavealpha( $destination, true );
            ::imagefill( $destination, 1, 1, $bgColor );

            ::imagecopyresampled(
                $destination,
                $this->image,
                0, 0,
                0, 0,
                $this->width, $this->height,
                $this->supersample( $this->width ), $this->supersample( $this-
>height )
            );
        } // ...
```

GD & PNG – Save (2)

```
        // Render depending on the chosen output type
        switch ( $this->type )
        {
            case IMG_PNG:
::imagepng( $destination, $file );
break;
            case IMG_JPEG:
::imagejpeg( $destination, $file, 100 );
break;
            default:
throw new Exception( "Unknown output type '{$this->type}'." );
        }
    }
```

Cairo & PNG – Initialize

```
protected function initialize()
{
    $this->surface = ::cairo_image_surface_create(
        ::CAIRO_FORMAT_ARGB32,
        $this->width,
        $this->height
    );

    $this->context = ::cairo_create( $this->surface );
    ::cairo_set_line_width( $this->context, 1 );
}
```

Cairo & PNG – Save

```
public function save( $file )
{
    ::cairo_surface_write_to_png( $this->surface, $file );
}
```

Ming & SWF – Initialize

```
protected function initialize()
{
    ::ming_setscale( 1.0 );
    $this->movie = new ::SWFMovie();
    $this->movie->setDimension(
        $this->modifyCoordinate( $this->width ),
        $this->modifyCoordinate( $this->height )
    );
    $this->movie->setRate( 1 );
    $this->movie->setBackground( 255, 255, 255 );
}

protected function modifyCoordinate( $coord )
{
    return $coord * 10;
}
```

- Ticks: http://www.half-serious.com/swf/format/basic-types/index.html#coordinates

Ming & SWF – Save

```
public function save( $file, $compression = 9 )
{
    $this->movie->save( $file, $compression );
}
```

Formats

- Libraries
- Basic datastructures
- Creating the surface
- First shape: polygon
- Gradients

The base class – extensions

```
    /**
     * Draws a single, optionally filled, polygon.
     *
     * As the first parameter the polygon expects an array with
Coordinate
     * objects, a color for the polygon and the fill status,
which defaults to
     * filled.
     *
     * @param array(Coordinate) $points
     * @param Color $color
     * @param boolean $filled
     */
    abstract public function drawPolygon( array $points, Color
$color, $filled = true );
```

DOM & SVG – set the style

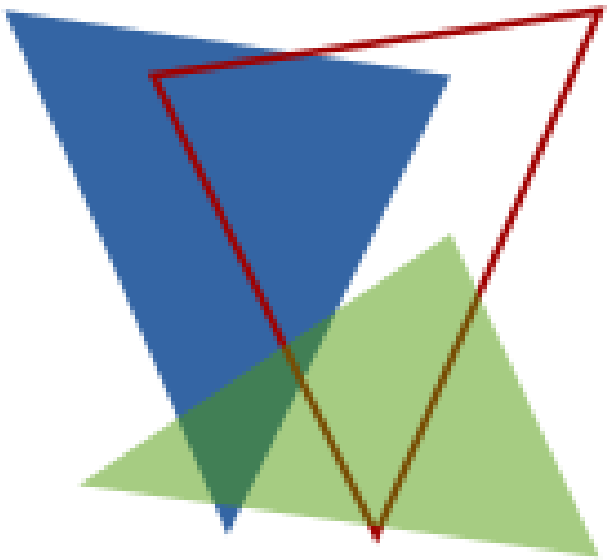
```
    protected function getStyle( Color $color, $filled )
    {
        if ( $filled )
        {
            return sprintf( 'fill: #%02x%02x%02x; fill-opacity: %.2f;
stroke: none;',
$color->red,
$color->green,
$color->blue,
1 - ( $color->alpha / 255 )
            );
        }
        else
        {
            return sprintf( 'fill: none; stroke: #%02x%02x%02x; stroke-
width: 1; stroke-opacity: %.2f;',
$color->red,
$color->green,
$color->blue,
1 - ( $color->alpha / 255 )
            );
        }
    }
```

DOM & SVG – Create a polygon

```
    public function drawPolygon( array $points, Color $color, $filled = true )
    {
        $lastPoint = end( $points );
        $pointString = sprintf( ' M %.4F,%.4F',
            $lastPoint->x, $lastPoint->y
        );

        foreach ( $points as $point )
        {
            $pointString .= sprintf( ' L %.4F,%.4F',
$point->x, $point->y
            );
        }
        $pointString .= ' z ';

        $path = $this->dom->createElement( 'path' );
        $path->setAttribute( 'd', $pointString );

        $path->setAttribute( 'style', $this->getStyle( $color, $filled ) );
        $path->setAttribute(
            'id', $this->elementPrefix . 'Polygon_' . ++$this->elementID
        );
        $this->elements->appendChild( $path );
    }
```

DOM & SVG – Draw!

```
$graphic = new Svg( 100, 100 );
$graphic->drawPolygon(
    array(
        new Coordinate( 10, 14 ),
        new Coordinate( 70, 23 ),
        new Coordinate( 40, 87 ),
    ), new Color( '#3465A4' )
);
$graphic->drawPolygon(
    array(
        new Coordinate( 90, 14 ),
        new Coordinate( 30, 23 ),
        new Coordinate( 60, 87 ),
    ), new Color( '#A00000' ), false
);
$graphic->drawPolygon(
    array(
        new Coordinate( 20, 80 ),
        new Coordinate( 70, 45 ),
        new Coordinate( 90, 90 ),
    ), new Color( '#4E9A067F' )
);
$graphic->save( 'images/example_svg_02.svg' );
```



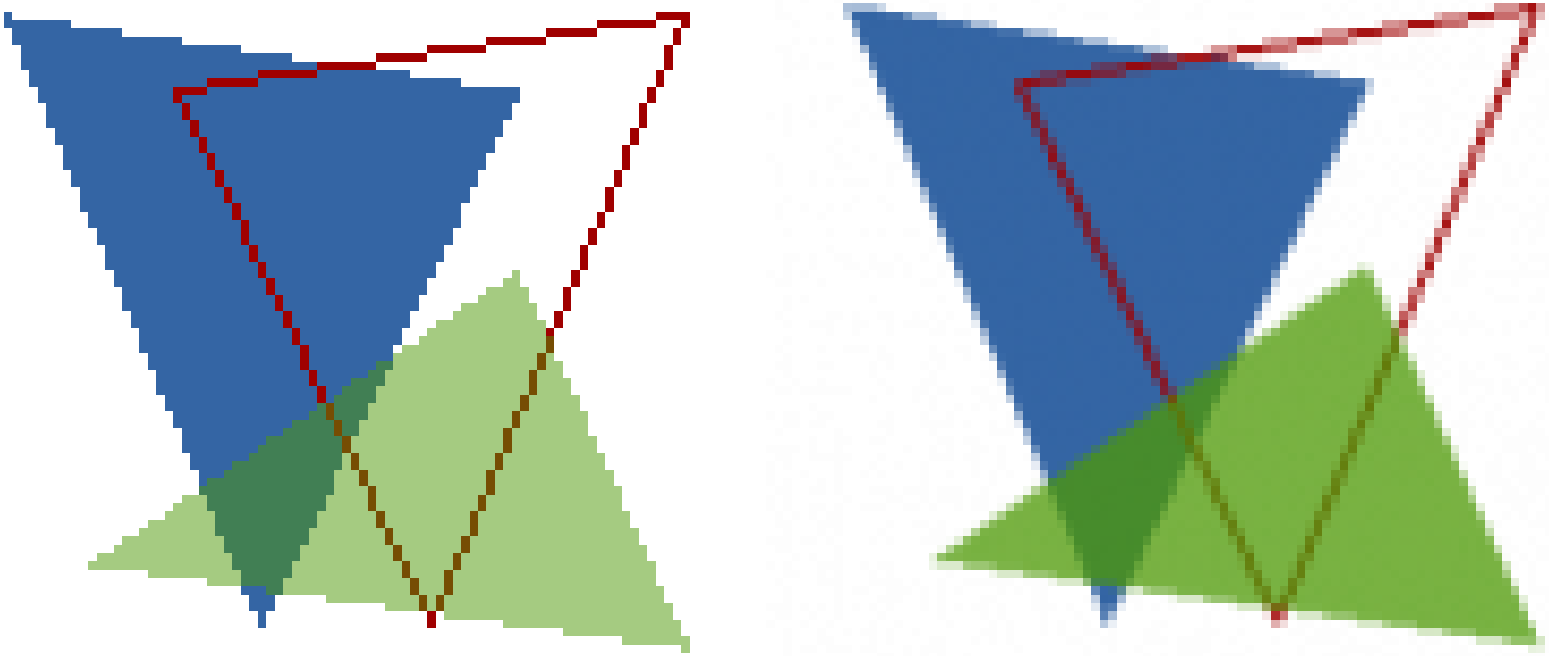
GD & PNG – Set the style

```
    protected function allocate( Color $color )
    {
        if ( $color->alpha > 0 )
        {
            return ::imagecolorallocatealpha( $this->image,
$color->red, $color->green, $color->blue, $color->alpha / 2 );
        }
        else
        {
            return ::imagecolorallocate( $this->image, $color-
>red, $color->green, $color->blue );
        }
    }
```

GD & PNG – Create a polygon

```
    public function drawPolygon( array $points, Color $color, $filled = true )
    {
        $drawColor = $this->allocate( $color );

        $pointArray = array();
        foreach( $points as $point )
        {
            $pointArray[] = $this->supersample( $point->x );
            $pointArray[] = $this->supersample( $point->y );
        }

        if ( $filled )
        {
            ::imagefilledpolygon( $this->image, $pointArray, count( $points ),
$drawColor );
        }
        else
        {
            ::imagepolygon( $this->image, $pointArray, count( $points ),
$drawColor );
        }

        return $points;
    }
```

GD & PNG – The result



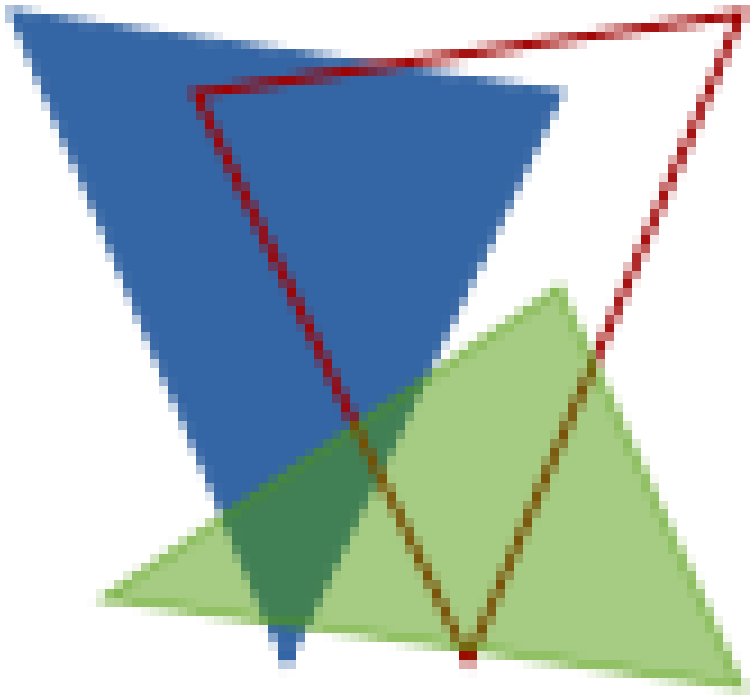
Cairo & PNG – Set the style

```
protected function setStyle( Color $color, $filled )
{
    ::cairo_set_source_rgba(
        $this->context,
        $color->red / 255,
        $color->green / 255,
        $color->blue / 255,
        1 - $color->alpha / 255
    );

    if ( $filled )
    {
        ::cairo_fill_preserve( $this->context );
    }
}
```

Cairo & PNG – Create a polygon

```
    public function drawPolygon( array $points, Color $color,
$filled = true )
    {
        $path = ::cairo_new_path( $this->context );

        $lastPoint = end( $points );
        ::cairo_move_to( $this->context, $lastPoint->x,
$lastPoint->y );

        foreach ( $points as $point )
        {
            ::cairo_line_to( $this->context, $point->x, $point-
>y );
        }

        ::cairo_close_path( $this->context );

        $this->setStyle( $color, $filled );
        ::cairo_stroke( $this->context );
    }
```

Cairo & PNG – The result



Ming & SWF – Set the style

```
    protected function setShapeStyle( SWFShape $shape, Color $color, $filled )
    {
        if ( $filled )
        {
            $fill = $shape->addFill(
$color->red,
$color->green,
$color->blue,
255 - $color->alpha
            );
            $shape->setLeftFill( $fill );
        }
        else
        {
            $shape->setLine(
$this->modifyCoordinate( 1 ),
$color->red,
$color->green,
$color->blue,
255 - $color->alpha
            );
        }
    }
```

Ming & SWF – Create a polygon

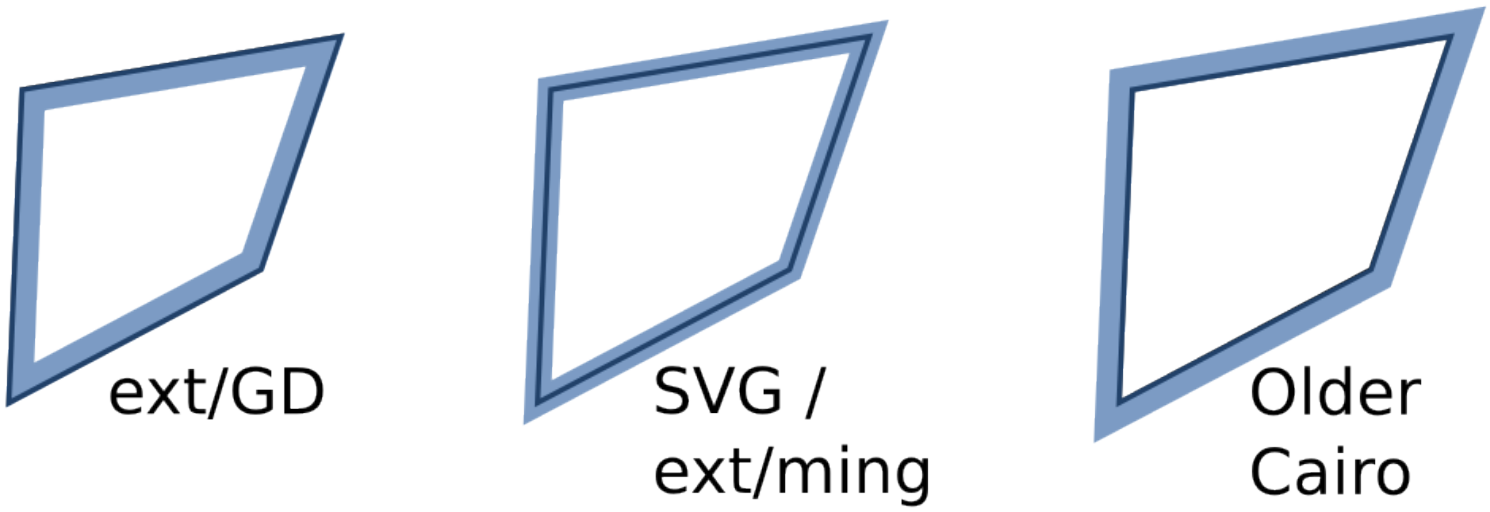
```
    public function drawPolygon( array $points, Color $color,
$filled = true )
    {
        $shape = new SWFShape();
        $this->setShapeStyle( $shape, $color, $filled );

        $lastPoint = end( $points );
        $shape->movePenTo(
            $this->modifyCoordinate( $lastPoint->x ),
            $this->modifyCoordinate( $lastPoint->y )
        );

        foreach ( $points as $point )
        {
            $shape->drawLineTo(
$this->modifyCoordinate( $point->x ),
$this->modifyCoordinate( $point->y )
            );
        }

        $object = $this->movie->add( $shape );
    }
```

Ming & SWF – The result

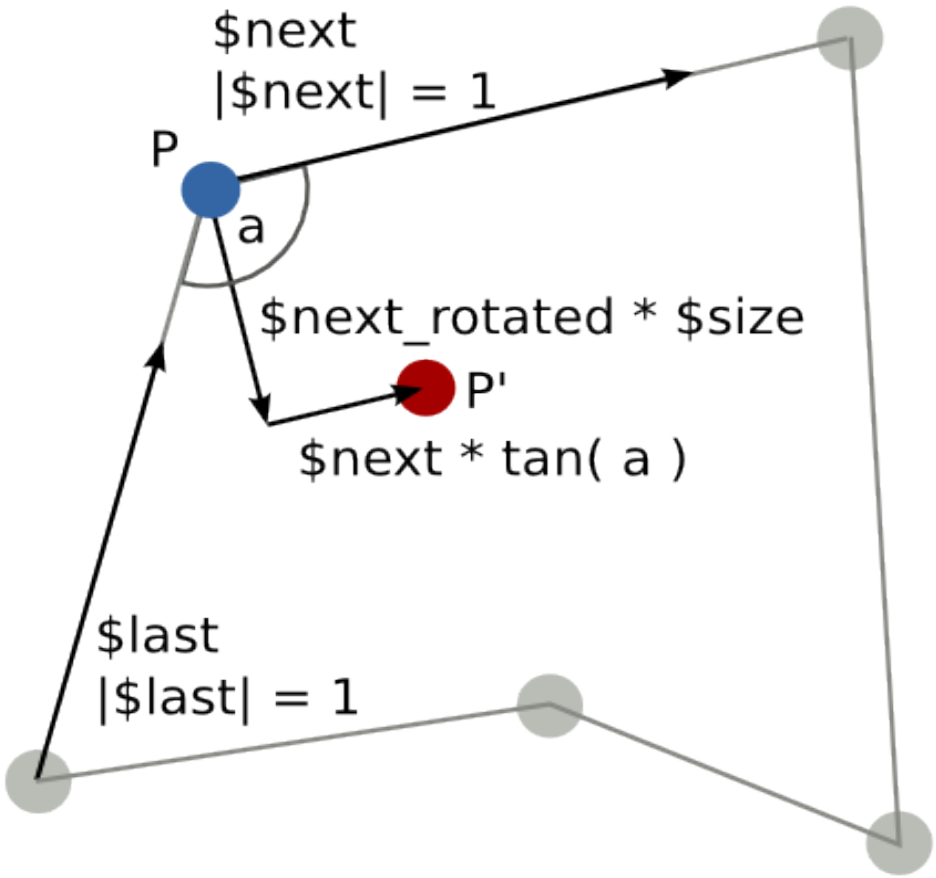
- This is flash ...
  - Open the zip file and view in your browser :)

Different border placements

- Not yet noticeable, but: Different backends place borders at different positions
  - User visible for big border widths
  - User visible for overlapping shapes



Different border placements – Theory



Different border placements – Usage

```
    public function drawPolygon( array $points, Color $color,
$filled = true )
    {
        // Fix polygone size for non filled polygones
        if ( !$filled )
        {
            $points = $this->reducePolygonSize( $points, .5 );
        }

        // The known code...
    }
```

Conclusion

- Even the simplest shape has problems when it comes to abstraction
  - Imagine border size reducement for circle sectors
- You may brake down most shapes to polygones
  - Circles, Rectangles, ...
- ezcGraph also implements circles, ellipse sectors and lines

Formats

- Libraries
- Basic datastructures
- Creating the surface
- First shape: polygon
- Gradients

Extend the color structure

```
class RadialGradient extends Color
{
    public function __construct( Color $startColor, Color $endColor,
Coordinate $center, $width, $height )
    {
        // Just set the properties
        $this->startColor = $startColor;
        $this->endColor = $endColor;
        $this->center = $center;
        $this->width = $width;
        $this->height = $height;

        // Fallback to average color of start end end color for
incopatible
        // backends
        foreach ( $startColor as $key => $value )
        {
            $this->$key = ( $value + $endColor->$key ) / 2;
        }
    }
}
```

Example implementation – Cairo & PNG (1)

```
    protected function setStyle( Color $color, $filled )
    {
        switch ( true )
        {
            case ( $color instanceof LinearGradient ):
// Handle other gradient types..
break;
            case $color instanceof RadialGradient:
// Our implementation ...
break;
            default:
::cairo_set_source_rgba(
    $this->context,
    $color->red / 255,
    $color->green / 255,
    $color->blue / 255,
    1 - $color->alpha / 255
);
break;
        }

        if ( $filled )
        {
            ::cairo_fill_preserve( $this->context );
        }
    }
```

Example implementation – Cairo & PNG (2)

```
$pattern = ::cairo_pattern_create_radial(
    0, 0, 0,
    0, 0, 1
);

::cairo_pattern_add_color_stop_rgba (
    $pattern,
    0,
    $color->startColor->red / 255,
    $color->startColor->green / 255,
    $color->startColor->blue / 255,
    1 - $color->startColor->alpha / 255
);

::cairo_pattern_add_color_stop_rgba (
    $pattern,
    1,
    $color->endColor->red / 255,
    $color->endColor->green / 255,
    $color->endColor->blue / 255,
    1 - $color->endColor->alpha / 255
);
```

Example implementation – Cairo & PNG (2)

```
// Scale pattern, and move it to the correct position
$matrix = cairo_matrix_multiply(
    $move = ::cairo_matrix_create_translate( -$color->center->x, -$color-
>center->y ),
    $scale = ::cairo_matrix_create_scale( 1 / $color->width, 1 / $color-
>height )
);
::cairo_pattern_set_matrix( $pattern, $matrix );

::cairo_set_source( $this->context, $pattern );
::cairo_fill( $this->context );
```

Cairo & PNG – Usage

```
$graphic = new Cairo( 150, 150 );
$graphic->drawPolygon(
    array(
        new Coordinate( 25, 60 ),
        new Coordinate( 95, 60 ),
        new Coordinate( 95, 130 ),
        new Coordinate( 25, 130 ),
    ),
    new RadialGradient(
        new Color( '#407cd2' ),
        new Color( '#245398' ),
        new Coordinate( 95, 60 ), 70, 70
    )
);
// ... add more shapes
$graphic->save( 'images/example_cairo_04.png' );
```

The other backends

- SVG
  - Full implementation possible, read the source.
- Ming
  - Gradients are nearly not documented
  - Hard to implement, but still works
- GD
  - No gradient support

The results

Questions?

- Open questions?
- Ressources:
  - http://kore-nordmann.de

The end

- Thanks for listening