Improving app_server for WebKit

Julian Harnath
<julian.harnath@rwth-aachen.de>
Introduction
Transparency Layers
Transforms and Clipping
Little Things Add Up
Outlook
Applications on Haiku

Server

app_server

Server link

Application

Clients
Applications on Haiku

Server

- app_server
- ServerApp
- ServerWindow
- BApplication
- BWindow

Clients

Server link

Interface Kit
Getting to Painter (1)

A very incomplete overview…

MessageLooper

ServerWindow

 Receives
app_server
messages
Getting to Painter (1)

A very incomplete overview…

MessageLooper

ServerWindow

Receives app_server messages

“Window on screen”

Canvas

State, coordinate conversion, …

DrawState

Entries of state stack

Window

View

OffscreenCanvas

Drawing into offscreen buffer (e.g. BView attached to BBitmap)
Getting to Painter (1)

A very incomplete overview…

**ServerWindow**
- Receives `app_server` messages
- "Window on screen"

**View**
- State, coordinate conversion, …

**Canvas**
- Entries of state stack
- Drawing into offscreen buffer (e.g. `BView` attached to `BBitmap`)

**MessageLooper**

**OffscreenCanvas**

**DrawingEngine**
Getting to Painter (1)

A very incomplete overview…

- **ServerWindow**
  - Receives app_server messages
  - “Window on screen”

- **MessageLooper**
  - Sends messages to **Canvas**

- **Canvas**
  - State, coordinate conversion, …

- **OffscreenCanvas**
  - Drawing into offscreen buffer (e.g. BView attached to BBitmap)
  - Entry of state stack

- **DrawState**
  - “Graphics card”
  - Frame buffer pointer, drawing using hardware (simple primitives), …

- **DrawingEngine**

- **Painter**
  - Draws things!
  - (in software)

- **HWInterface**
  - “Graphics card”
  - Frame buffer pointer, drawing using hardware (simple primitives), …

- **AGG**
Getting to Painter (2)

Drawing a rectangle…

```cpp
BView::FillRect()
```

Client

app_server
Drawing a rectangle…

Client

app_server

Message: AS_FILL_RECT with BRect payload

ServerWindow

BView::FillRect()
Getting to Painter (2)

Drawing a rectangle...

Client → Message: AS_FILL_RECT with BRect payload → app_server → ServerWindow

View::PenToScreenTransform() → Transform rect to screen coordinates

Window::GetDrawingEngine() → Get the engine of this window
Getting to Painter (2)

Drawing a rectangle…

Client
---
app_server

BView::FillRect()

Message: AS_FILL_RECT with BRect payload

ServerWindow

View::PenToScreenTransform()

Transform rect to screen coordinates

Window::GetDrawingEngine()

Get the engine of this window

DrawingEngine::FillRect()

Painter::ClipRect()

State? HW accel?

Painter::FillRect()

HWInterface::FillRegion()
Transparency Layers

α = 0.7

Global alpha

Transparency layer
The Workaround

(1) Create a new empty BBitmap and attach BView

(2) Draw into bitmap

(3) Draw bitmap onto background with added transparency (via ClipToPicture), throw away BBitmap
The Problem

(1) Create a new empty BBitmap and attach BView

On some websites, WebKit likes to use many layers, especially when doing many renders during scrolling.

**Bad**: we don’t know the size of the drawing yet, so we have to create the BBitmap at view size. In WebPositive, this is almost the whole browser window size!

**Worse**: attaching a BView to a BBitmap spawns an offscreen window thread inside app_server.
A Better Solution

Let app_server know what we’re doing!
A Better Solution

Let app_server know what we’re doing!

1. Draw circle
2. Draw rectangle
3. Draw triangle

(1) Client: start layer; then draw things.
app_server does not draw, and instead just writes down the list of operations.
A Better Solution

Let app_server know what we’re doing!

1. Draw circle
2. Draw rectangle
3. Draw triangle

(1) Client: start layer; then draw things. app_server does not draw, and instead just writes down the list of operations.

(2) Client: end layer. app_server looks at the operations written down and figures out the (approx.) bounding box of this drawing, without actually drawing it
A Better Solution

Let app_server know what we’re doing!

1. Draw circle
2. Draw rectangle
3. Draw triangle

(1) Client: start layer; then draw things. app_server does not draw, and instead just writes down the list of operations.

(2) Client: end layer. app_server looks at the operations written down and figures out the (approx.) bounding box of this drawing, without actually drawing it.

(3) Create UtilityBitmap of bounding box size and clear it. UtilityBitmap is app_server-internal and spawns no new thread!
A Better Solution

Let app_server know what we’re doing!

(1) Client: start layer; then draw things. app_server does not draw, and instead just writes down the list of operations.

(2) Client: end layer. app_server looks at the operations written down and figures out the (approx.) bounding box of this drawing, without actually drawing it.

(3) Create UtilityBitmap of bounding box size and clear it. UtilityBitmap is app_server-internal and spawns no new thread!

(4) Draw into UtilityBitmap from written down operations
A Better Solution

Let app_server know what we’re doing!

1. Draw circle
2. Draw rectangle
3. Draw triangle

(1) Client: start layer; then draw things. app_server does not draw, and instead just writes down the list of operations.

(2) Client: end layer. app_server looks at the operations written down and figures out the (approx.) bounding box of this drawing, without actually drawing it.

(3) Create UtilityBitmap of bounding box size and clear it. UtilityBitmap is app_server-internal and spawns no new thread!

(4) Draw into UtilityBitmap from written down operations

(5) Draw bitmap with transparency (via AlphaMask) and discard it
A Better Solution

Let app_server know what we’re doing!

(1) Client: start layer; then draw things.
   app_server does not draw, and instead just writes down the list of operations.

(2) Client: end layer.
   app_server looks at the operations written down and figures out the (approx.) bounding box of this drawing, without actually drawing it

(3) Create UtilityBitmap of bounding box size and clear it. UtilityBitmap is app_server-internal and spawns no new thread!

(4) Draw into UtilityBitmap from written down operations

(5) Draw bitmap with transparency (via AlphaMask) and discard it

Problem solved!
BPicture Saves the Day

1. Draw circle
2. Draw rectangle
3. Draw triangle

Details to observe: drawing offset, transforms, clipping, draw state, drawing mode, …
Layer API

void BView::BeginLayer(uint8 opacity);
void BView::EndLayer();
Transforms: R5

void BView::SetOrigin(float x, float y);
void BView::SetScale(float ratio);

Translation, Scaling
void BView::SetTransform(BAffineTransform transform);
BAffineTransform BView::Transform() const;

BAffineTransform (Haiku)

Translation, Scaling/Mirroring, Rotation, Shearing

\[
\begin{pmatrix}
  p'_x \\
p'_y \\
1
\end{pmatrix} = \begin{pmatrix}
  s_x & s_{hx} & t_x \\
  s_{hy} & s_y & t_y \\
0 & 0 & 1
\end{pmatrix} \cdot \begin{pmatrix}
p_x \\
p_y \\
1
\end{pmatrix}
\]

BAffineTransform:: ...  
  AffineTranslation()  
  AffineRotation()  
  AffineScaling()  
  AffineShearing()
BAffineTransform (Haiku)

```cpp
void BView::SetTransform(BAffineTransform transform);
BAffineTransform BView::Transform() const;
```

Translation, Scaling/Mirroring, Rotation, Shearing

\[
\begin{pmatrix}
  s_x & sh_x & t_x \\
  sh_y & s_y & t_y \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  p_x \\
  p_y \\
  1
\end{pmatrix}
= \begin{pmatrix}
  p'_x \\
  p'_y \\
  1
\end{pmatrix}
\]

Composition

\[
\vec{p}' = B(A\vec{p}) = (BA)\vec{p}
\]

BAffineTransform:: ... 
AffineTranslation()
AffineRotation()
AffineScaling()
AffineShearing()

BAffineTransform:: ... 
Multiply(), PreMultiply()
TranslateBy(), ScaleBy(), RotateBy(), ShearBy()
...
Clipping

R5 API: clipping region

```c
void BView::ConstrainClippingRegion(BRegion* region);
```

*fast*

R5 API: clipping mask

```c
void BView::ClipToPicture(BPicture* picture, [...]);
void BView::ClipToInversePicture(BPicture* picture, [...]);
```

R5: 1 bit alpha – only fully opaque or fully transparent
Haiku: 8 bit alpha – allows full pixel alpha masking
Clipping Examples

No clipping
Clipping Region

Original  ConstrainClippingRegion()
Clipping Mask

Original

BPicture (Alpha Mask)

ClipToPicture()
Transforms and Clipping

When recalculating the clipping region, the R5 transforms are applied…
When recalculating the clipping region, the R5 transforms are applied...

...however, clipping regions know nothing about affine transforms.
R5 transforms...

SetScale()

ConstrainClippingRegion()
BAffineTransform...

ConstrainClippingRegion()

SetTransform()
BAffineTransform…

SetTransform()

SetTransform() + ConstrainClippingRegion()
Clipping Wish List

Fast for BRegion clipping

Allow complex clipping shapes

Aware of affine transformations

Always intersecting, no state push required
RFC: New Clipping API

void BView::ClipToRect(BRect rect);
void BView::ClipToInverseRect(BRect rect);

void BView::ClipToShape(BShape* shape);
void BView::ClipToInverseShape(BShape* shape);

• Works with affine transforms
• Automatically selects between fast region clipping and alpha masks (prefers region when possible)
• Directly clip to shape without needing BPicture
• “Inverse” variants to clip out
• Always intersecting, no state push required
• All variants can be freely mixed
• Now independent of view size, view origin change only reattaches buffer – no rerendering!
• Stacked alpha masks intersect their bounding boxes
Clipping and BAffineTransform

SetTransform()

SetTransform() + ClipToRect()
Factored out class BitmapPainter and classes for optimized variants

New optimized paths for:

- unscaled B_OP_COPY with alpha mask
- bilinear with pixel alpha overlay
Little Things (2): Drawing Modes

PixelFormat selects function pointers for blending pixels/lines/spans based on drawing mode, fill pattern, …

Added new functions for pixel alpha composite with solid fill color (no pattern)…

… considerable speedup
Little Things Can Be Hard To Find

Performance can be non-intuitive – always measure
Demo time!
Some Ideas

• Pre-multiplied alpha
• SIMD
• Cache for alpha masks
• Cache for scaled bitmaps
• Refactoring
  • Extract more things from Painter
  • Unify clipping, transforms
• Unit tests!
Thank you!

Questions?