GDC 13 EUROPE

# Broken Age's Approach to Scalability

**Oliver Franzke**
Lead Programmer, Double Fine Productions

GAME DEVELOPERS CONFERENCE EUROPE
COLOGNE, GERMANY
AUGUST 19-21, 2013
EXPO DATES: AUGUST 19-20
2013

# Content

- Introduction
- Platform diversity
- Game assets
  - Characters
  - Environments
  - Shaders

# Who am I?

- Lead Programmer of Broken Age
- Joined games industry in 2000
- Programming for 25 years
- p1xelcoder

# What is 

?

- Classic adventure game by Tim Schafer
- Modern look
- Funded with KICKSTARTER
  - 834% of original funding goal
  - We have awesome fans!
- Available on nearly everything*

* See „Platforms“ slide


# Trailer Time!

http://www.youtube.com/watch?v=BWM4R5JsakE

# The look of Broken Age

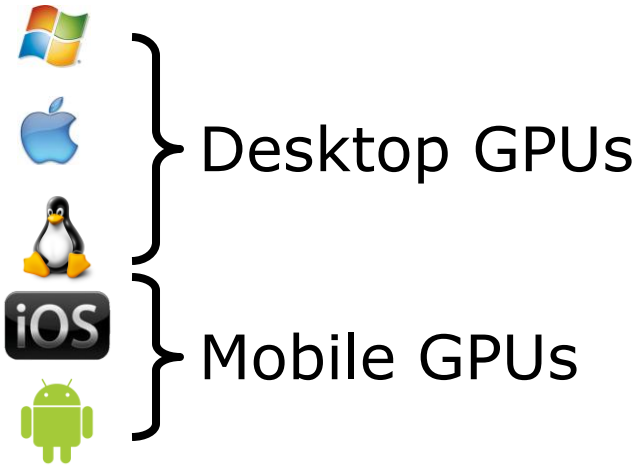
- Modern HD graphics
  - Expressive characters
  - Beautiful lively environments
  - Parallaxing (2.5D)
  - Dynamic lighting
  - Real-time reflections
  - 2D character shadows

# The look of Broken Age

- Typical scenes

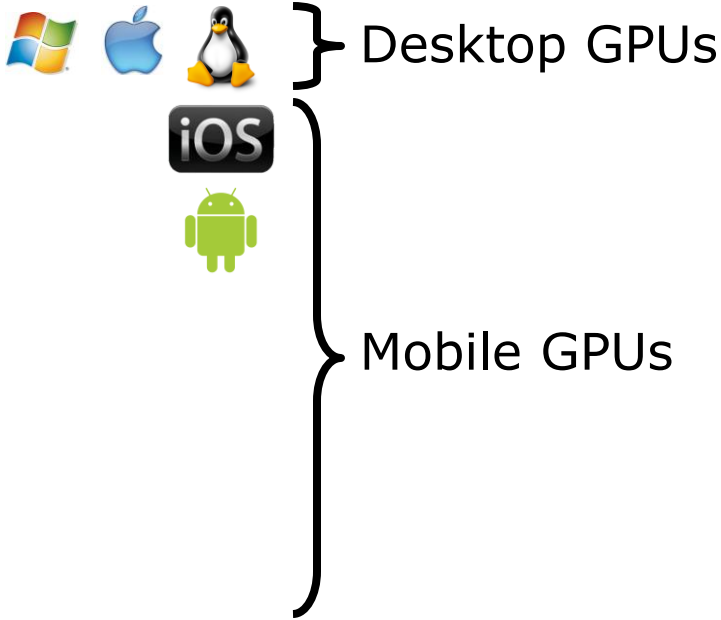

# Platforms, platforms, platforms!

- Optimistic view: 5 platforms
  - Windows
  - OSX
  - Linux
  - iOS mobile & tablets
  - Android mobile & tablets



# Platforms, platforms, platforms!

- Pessimistic view: 8+ platforms!
  - Windows, OSX, Linux
  - iOS mobile & tablets
  - Android mobile & tablets
    - PowerVR
    - NVIDIA
    - Qualcomm
    - Vanilla Android
    - Derivates (e.g. OUYA)



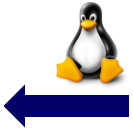
# Why is this a problem?

- GPUs are very different
  - Architectural goals
  - Supported features
  - Performance characteristics
- Platform fragmentation
  - No exact knowledge of device capabilities
  - Android fragmentation report:
    - http://opensignal.com/reports/fragmentation-2013/

# Scalability goals

- Amazing visuals on high-end PCs
  - High resolution
  - Fast
- Scale to low-end handheld devices
  - Low resolution
  - Low memory
  - Slow

# Scalability goals

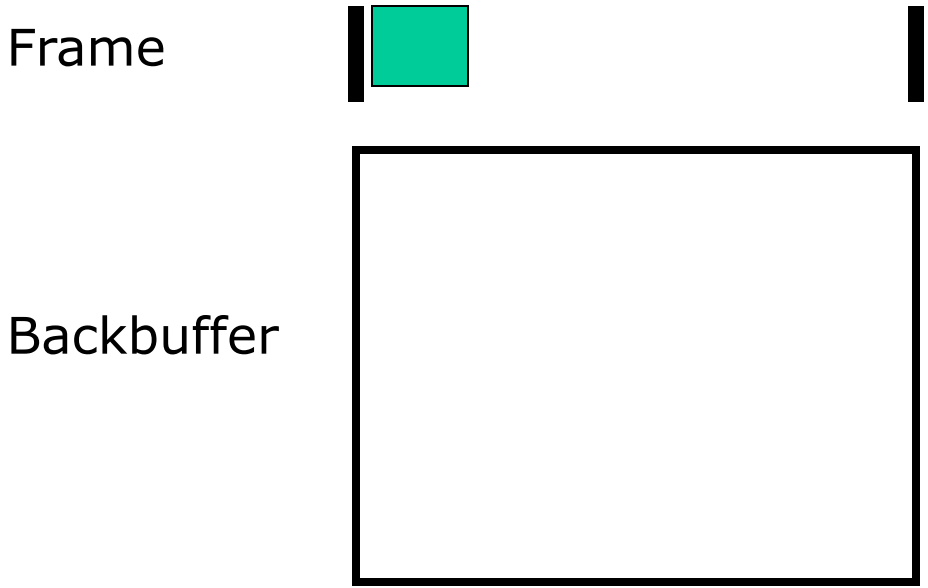


# A brief look at GPU types

- Desktop GPUs
  - Maximize throughput
  - Lots of memory
  - High power consumption (100W+)
  - No restrictions on draw calls
  - Transparency is not a problem

# A brief look at GPU types

- Desktop GPUs

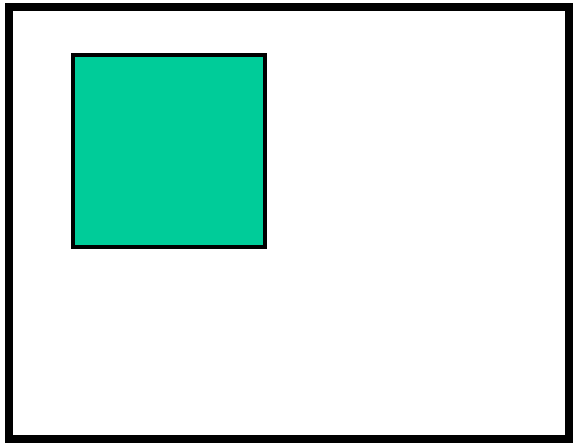


# A brief look at GPU types

- Desktop GPUs

Frame

Backbuffer

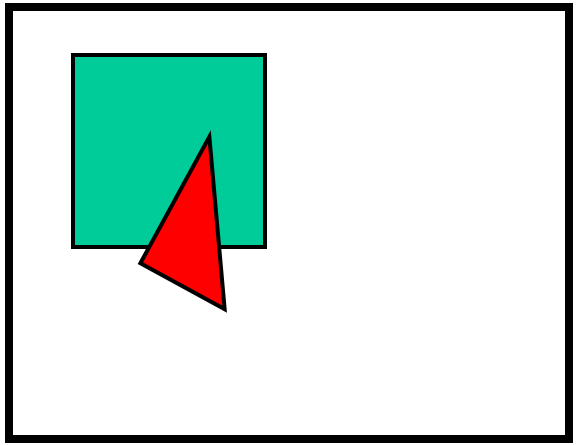


# A brief look at GPU types

- Desktop GPUs

Frame

Backbuffer

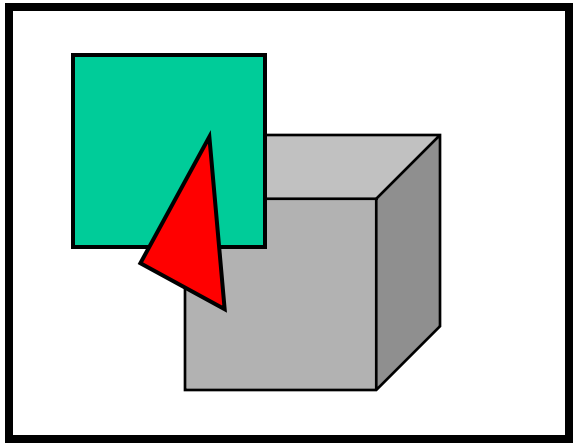


# A brief look at GPU types

- Desktop GPUs

Frame

Backbuffer

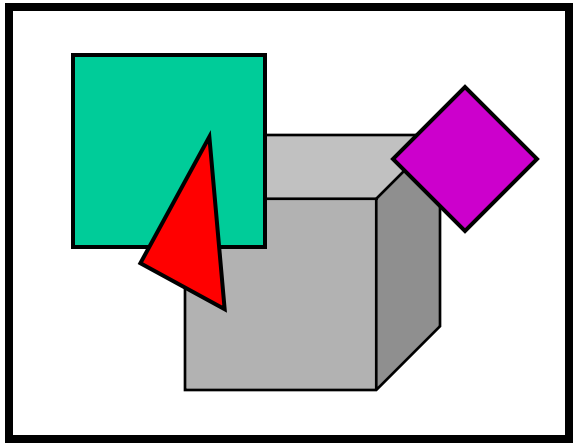


# A brief look at GPU types

- Desktop GPUs

Frame

Backbuffer

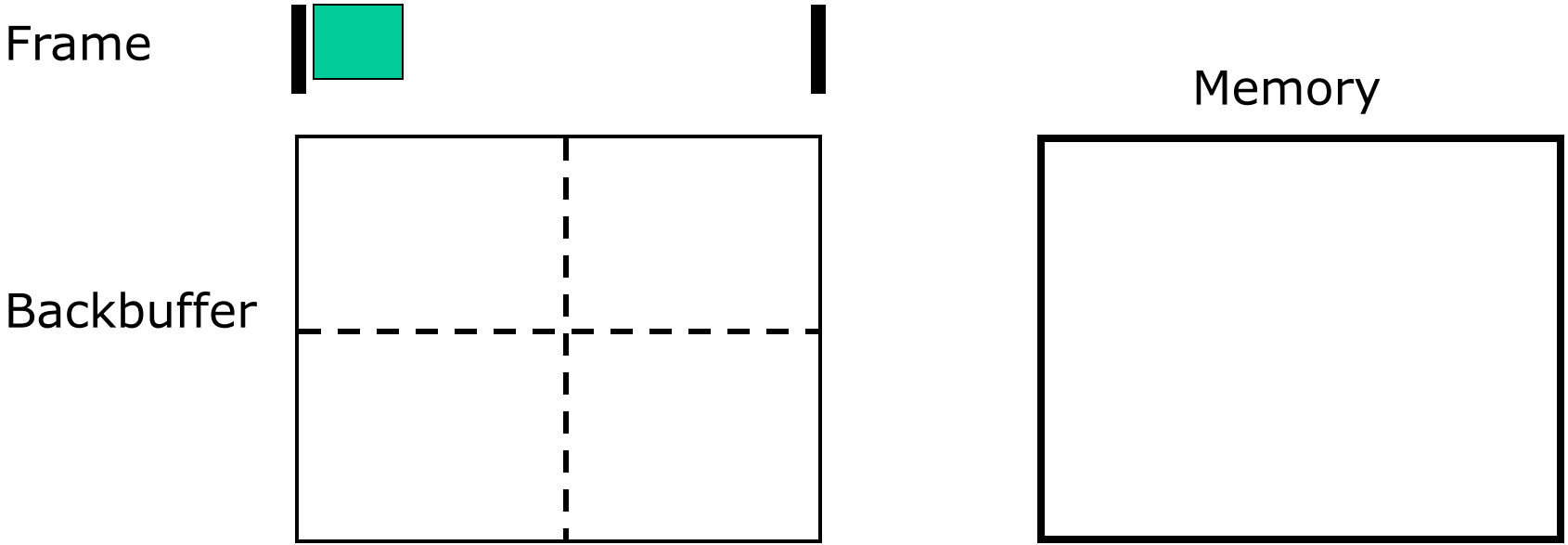


# A brief look at GPU types

- Mobile GPUs
  - Low power consumption (~100mW)
  - Limited memory
  - Tiled rendering architecture
  - Limitation to draw calls
  - Optimized for opaque geometry
  - Overdraw is expensive

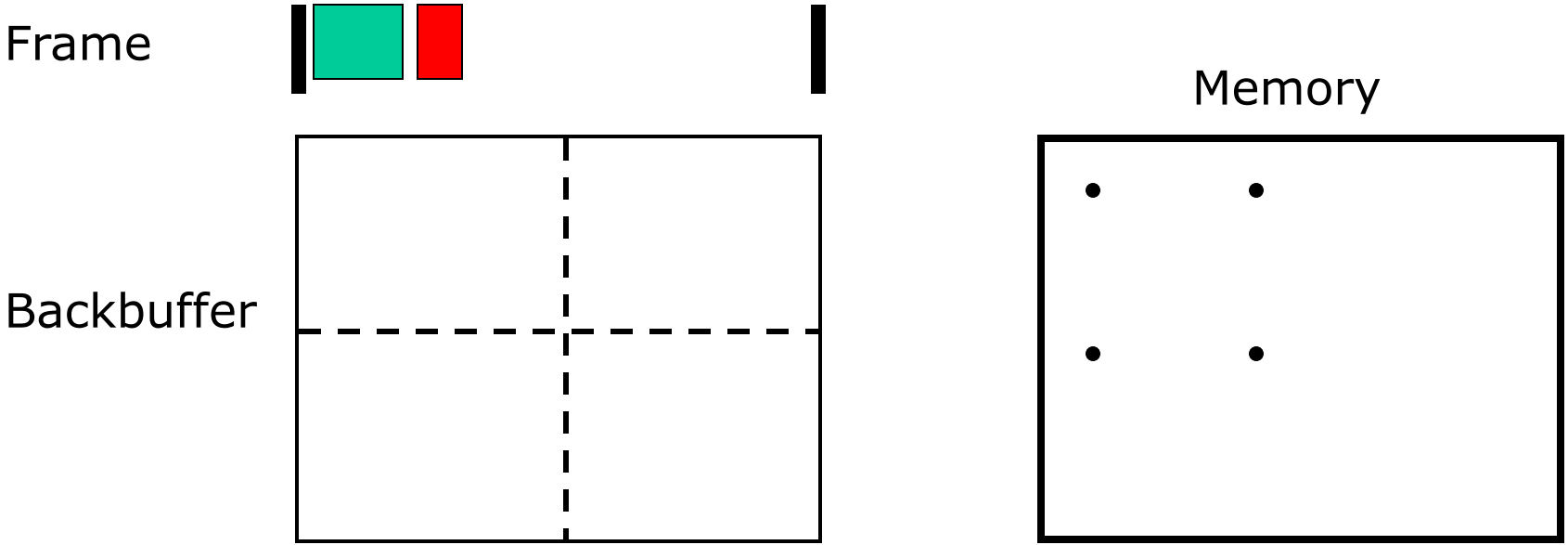
# A brief look at GPU types

- Mobile GPUs



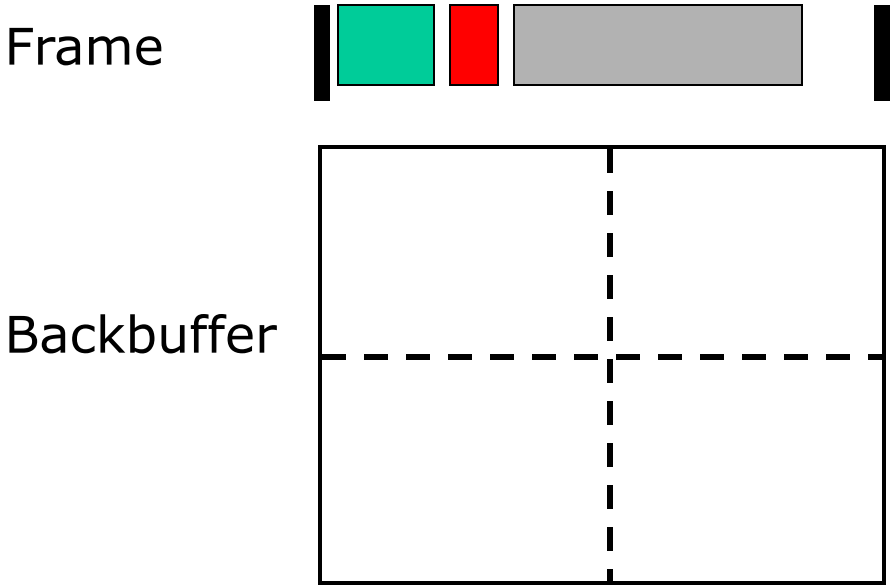
# A brief look at GPU types

- Mobile GPUs



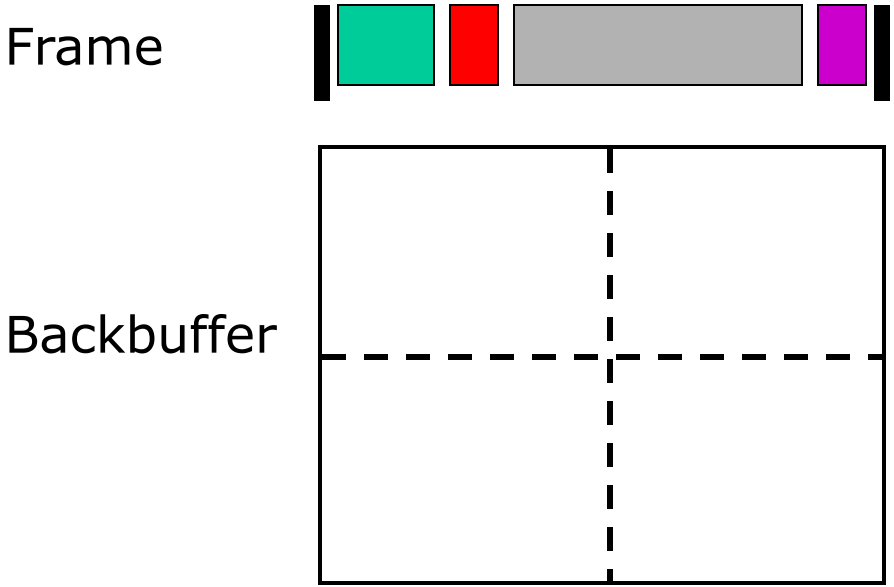
# A brief look at GPU types

- Mobile GPUs



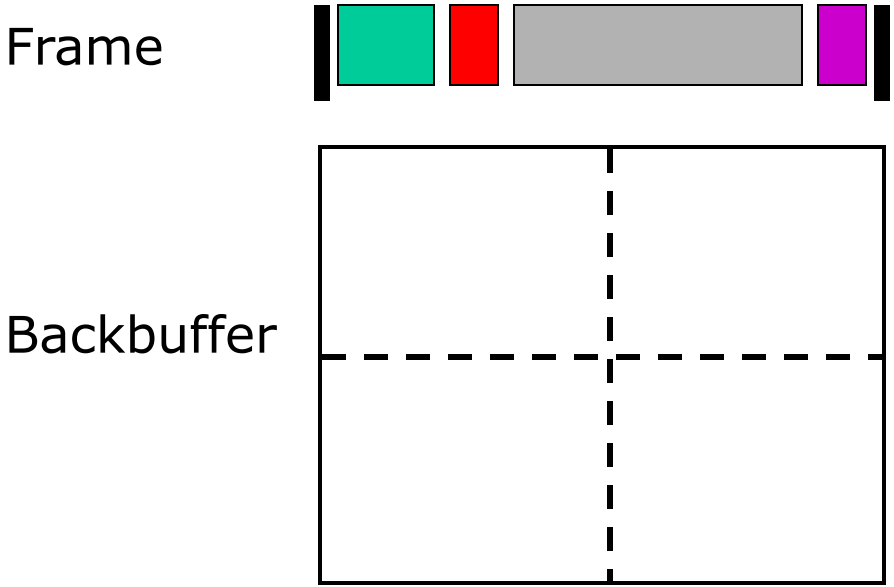
# A brief look at GPU types

- Mobile GPUs



# A brief look at GPU types

- Mobile GPUs

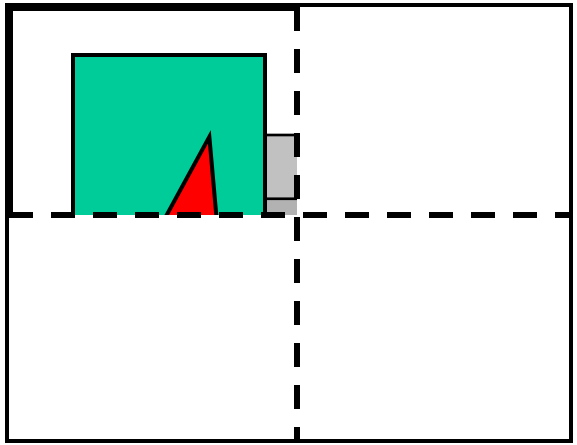


# A brief look at GPU types

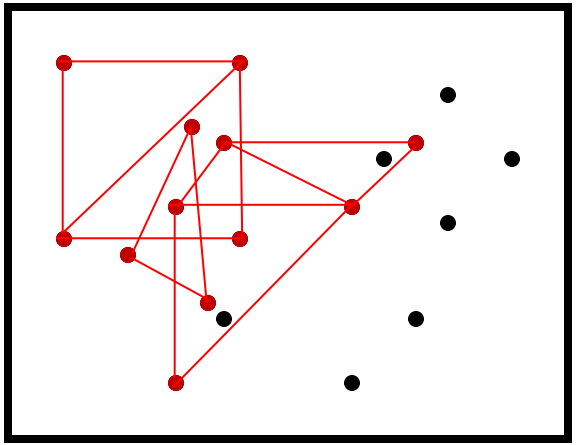
- Mobile GPUs

Frame

Backbuffer



Memory

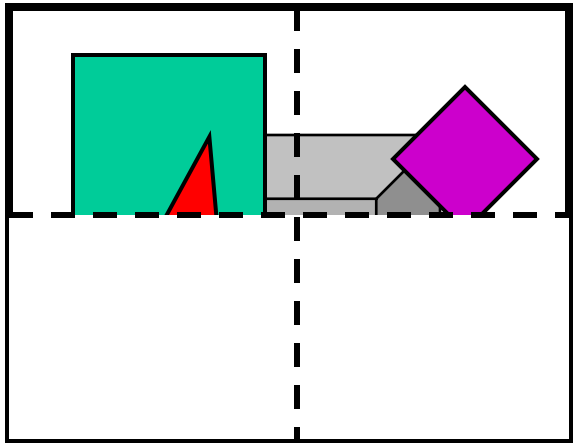


# A brief look at GPU types

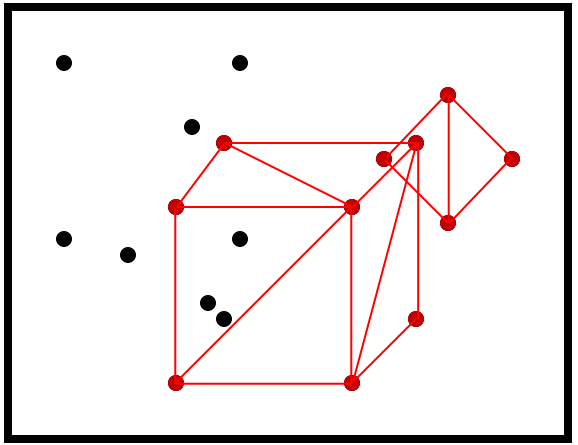
- Mobile GPUs

Frame

Backbuffer



Memory

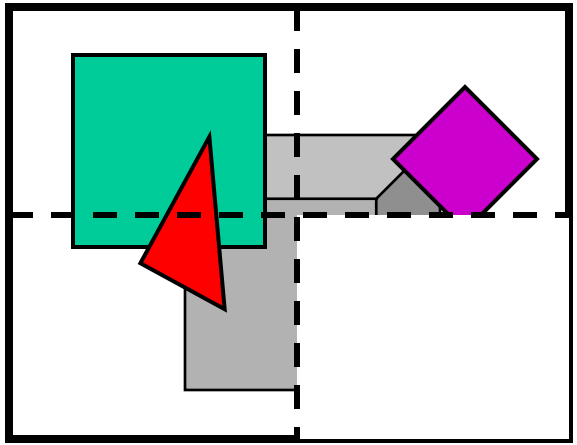


# A brief look at GPU types

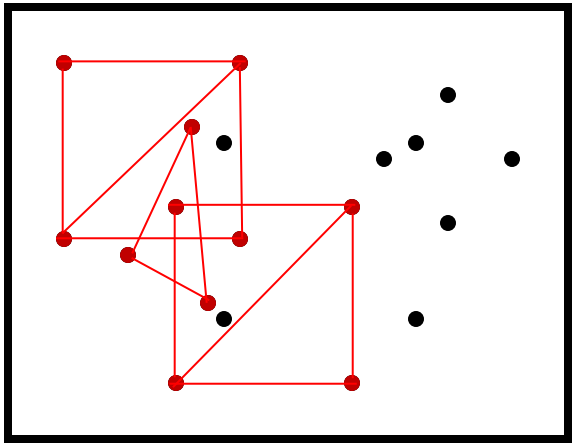
- Mobile GPUs

Frame

Backbuffer



Memory

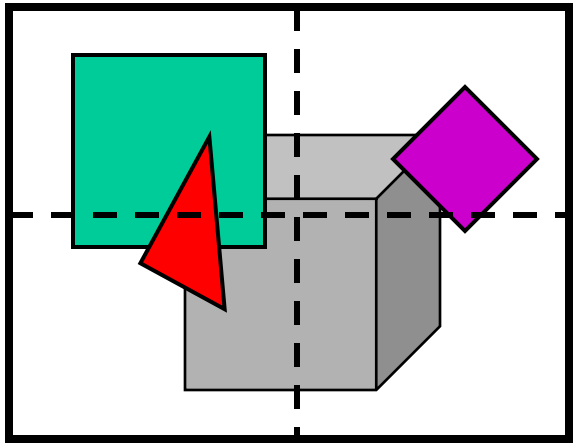


# A brief look at GPU types

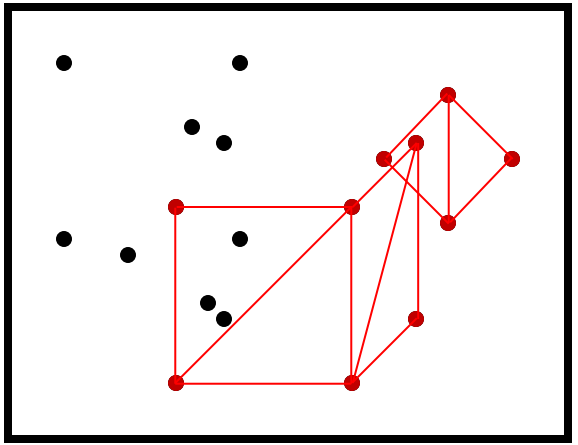
- Mobile GPUs

Frame

Backbuffer



Memory



# Consequences for Broken Age

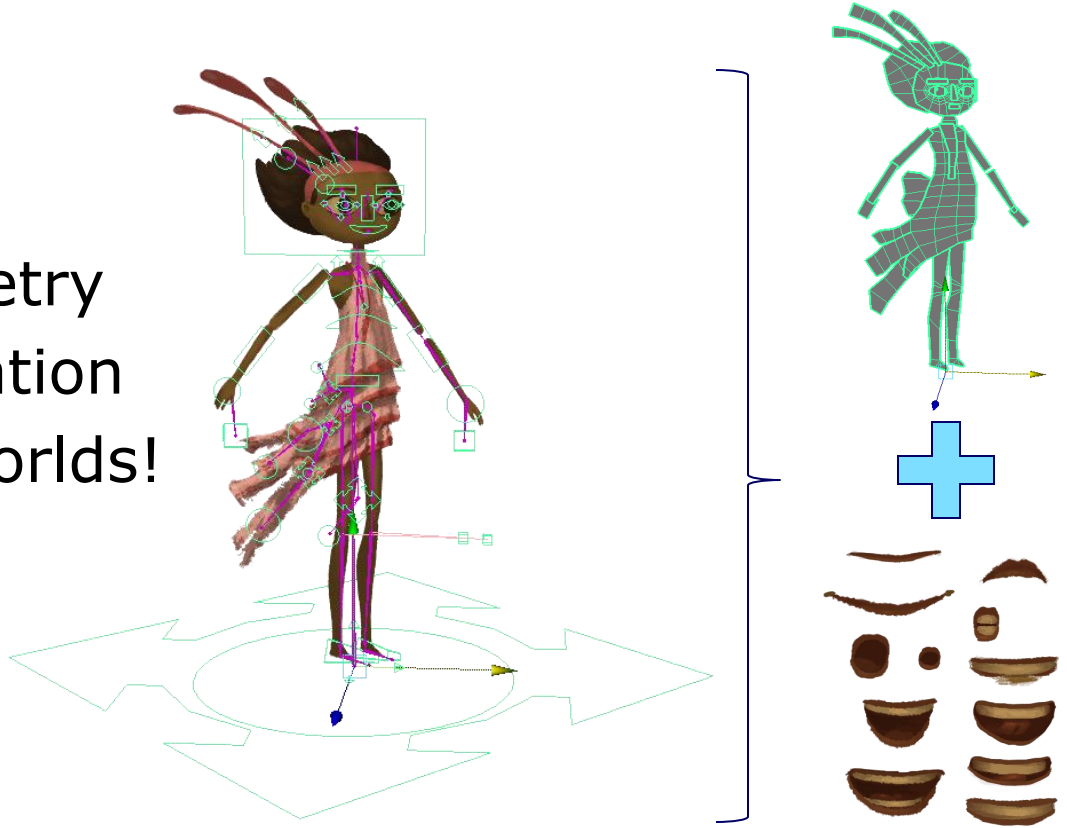
- Minimize overdraw
- Keep number of draw calls low (<100)
- No render-targets
- Avoid dependent texture look-ups
- Optimize shaders

# Characters

- Goals
  - Expressive
  - Efficient to animate
  - Flexible and extendible
  - Receive dynamic lighting
  - Minimize memory and overdraw
  - Use studio expertise

# Characters

- Hybrid rig
  - Skinned geometry
  - Flipbook animation
  - Best of both worlds!

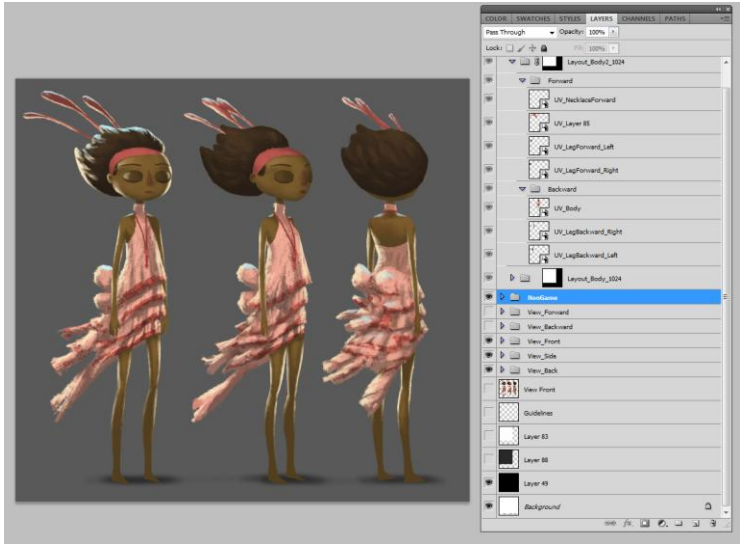


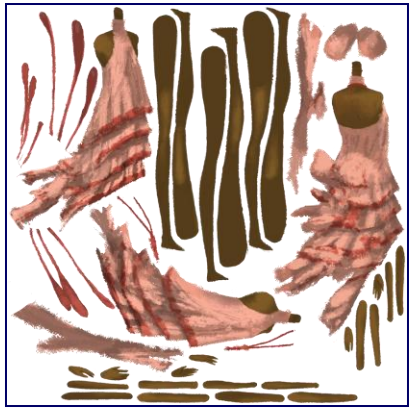
# Characters

- Hybrid rig – Authoring
  - Concept

# Characters

- Hybrid rig – Authoring
  - Texture layout

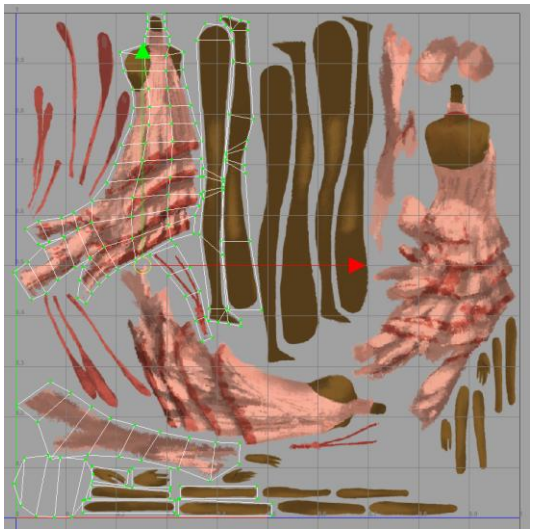




# Characters

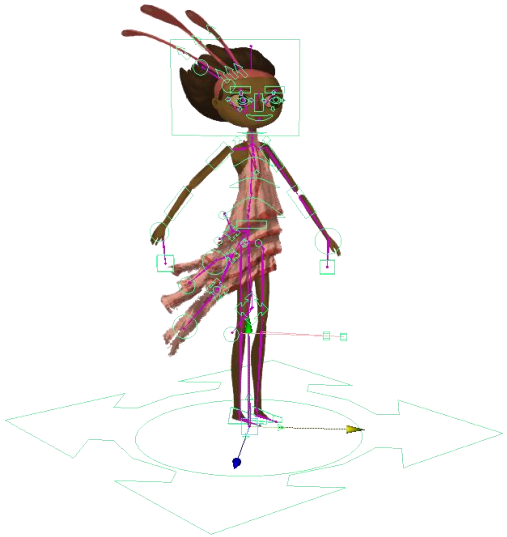
- Hybrid rig – Authoring
  - Skinned geometry for body parts

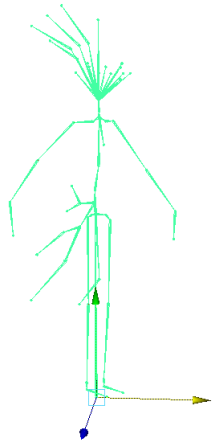





# Characters

- Hybrid rig – Authoring
  - One skeletal rig for all views!



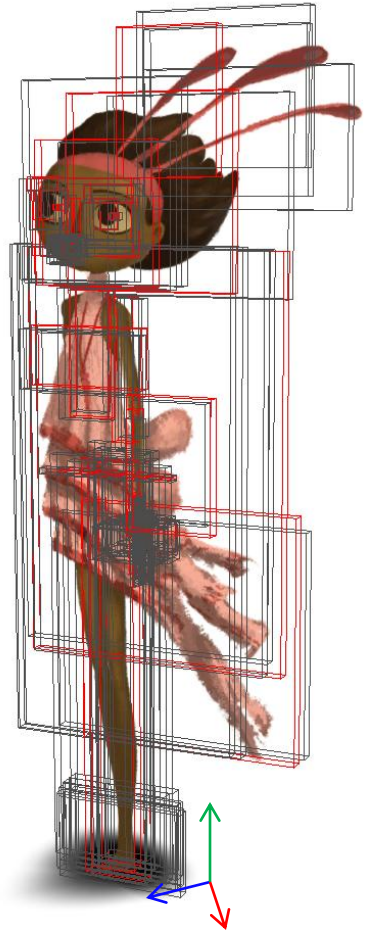


# Characters

- Hybrid rig – Authoring
  - Animation
    - Joint transform
    - Geometry visibility
  - Auto lip-sync generation
    - Annosoft lipsync library

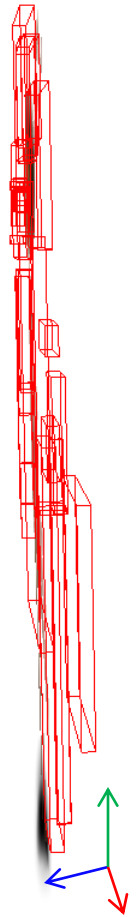
# Characters

- Hybrid rig – Run-time
  - Evaluate joint transforms subset visibility
  - Gather visible body parts



# Characters

- Hybrid rig – Run-time
  - Sort geometry
    - Back-to-front using AABBs
  - Batch draw calls



# Characters

- Hybrid rig vs. Flipbooks
  - More efficient creation
  - Repurpose existing tools
  - Lower memory footprint
    - Boy: ~23MB vs. ~5.5GB (1 : 244)

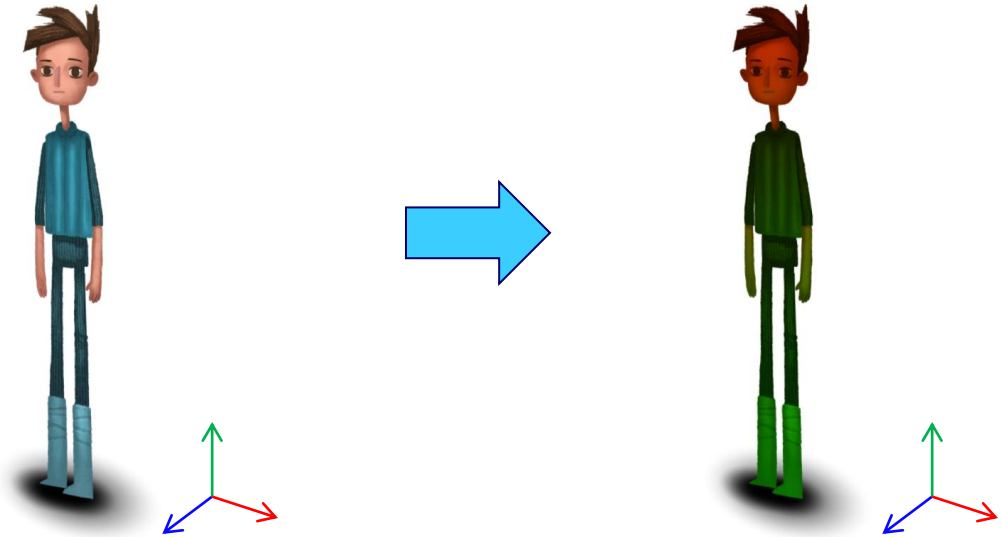
Hybrid rig + all animations:
5.8MB = 68810 frames (215 anims)
150KB = rig
17.4MB = textures (DXT5)

Flipbook estimation:
Frames = 34405 (anims @ 15fps)
Sprite size = 256 x 512 (DXT5)

Disclaimer: Pre-alpha values

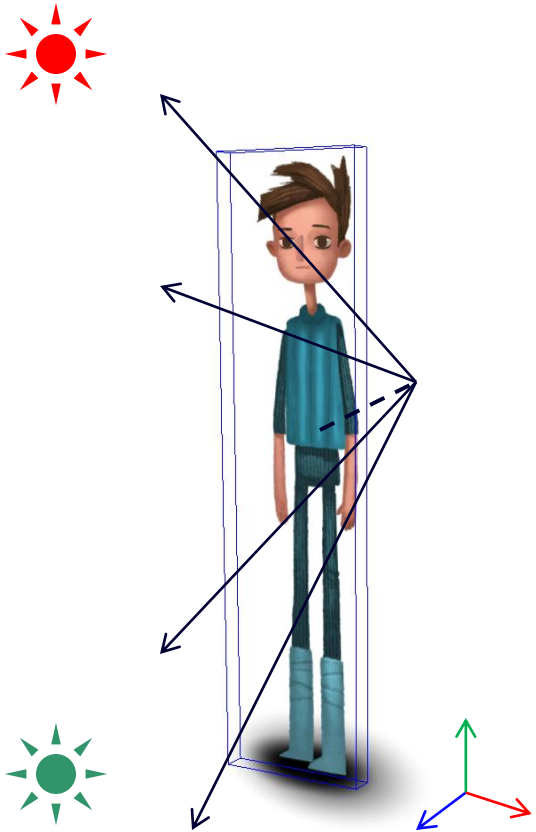
# Characters

- Lighting – Gradient lighting
  - Low-frequency lighting



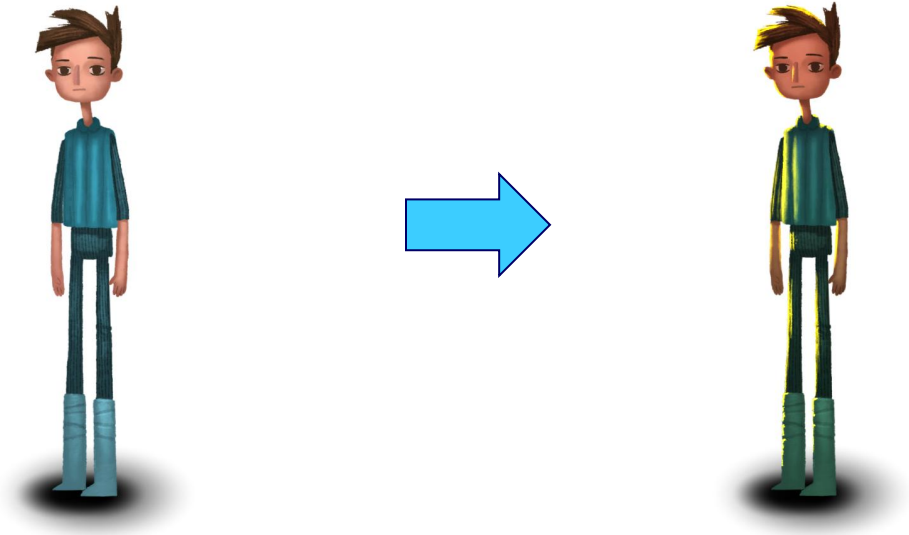
# Characters

- Lighting – Gradient lighting
  - Sample nearby sources
  - Average top and bottom color
  - Approximated normal
  - Cheap!



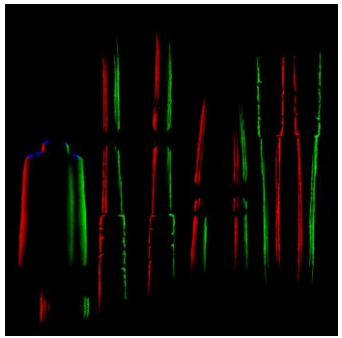
# Characters

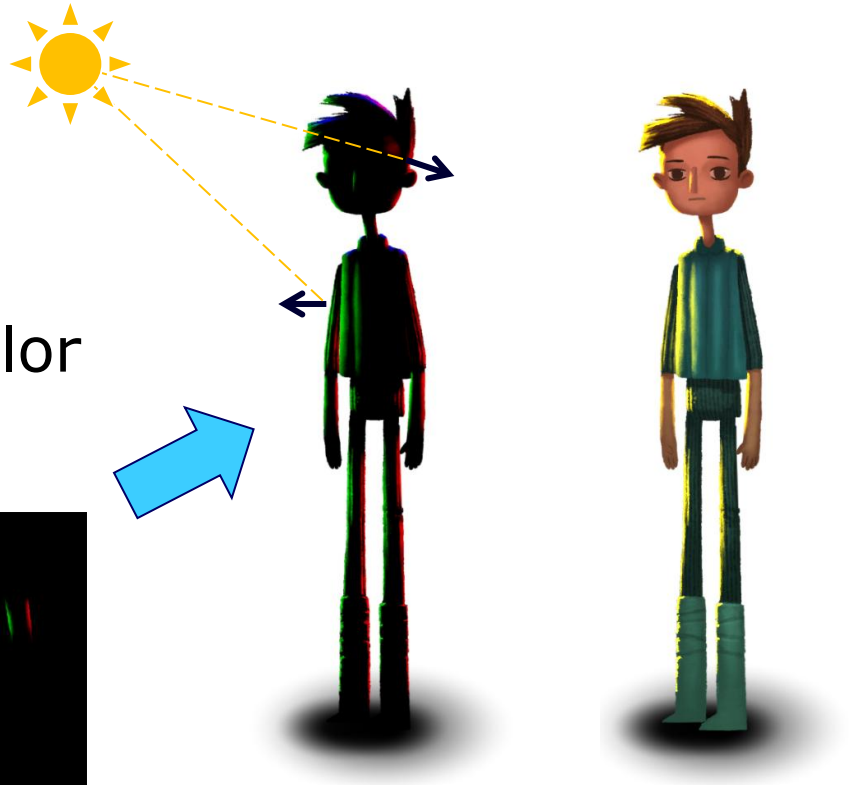
- Lighting – Rim lighting
  - Edge highlighting



# Characters

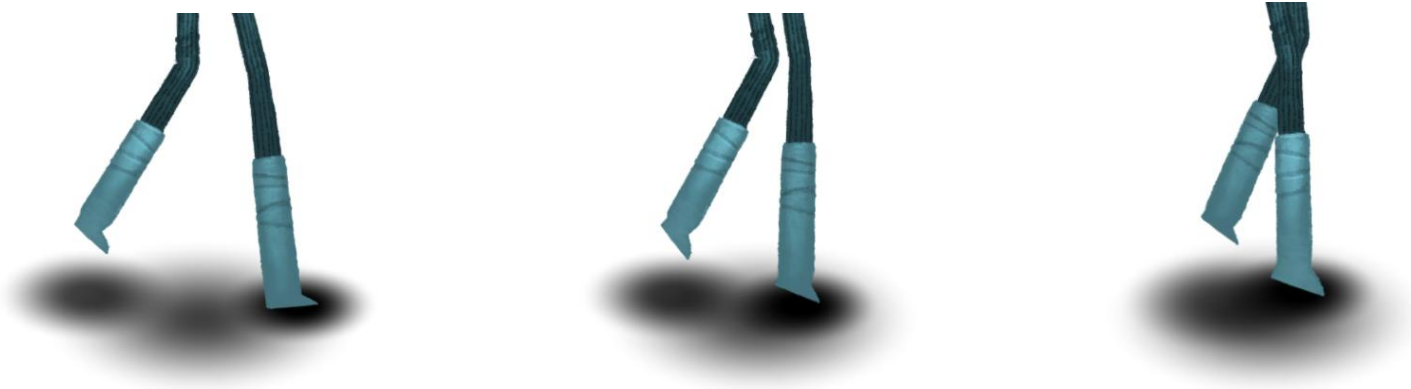
- Lighting – Rim lighting
  - Local space normal map
  - Average direction and color
  - Expensive





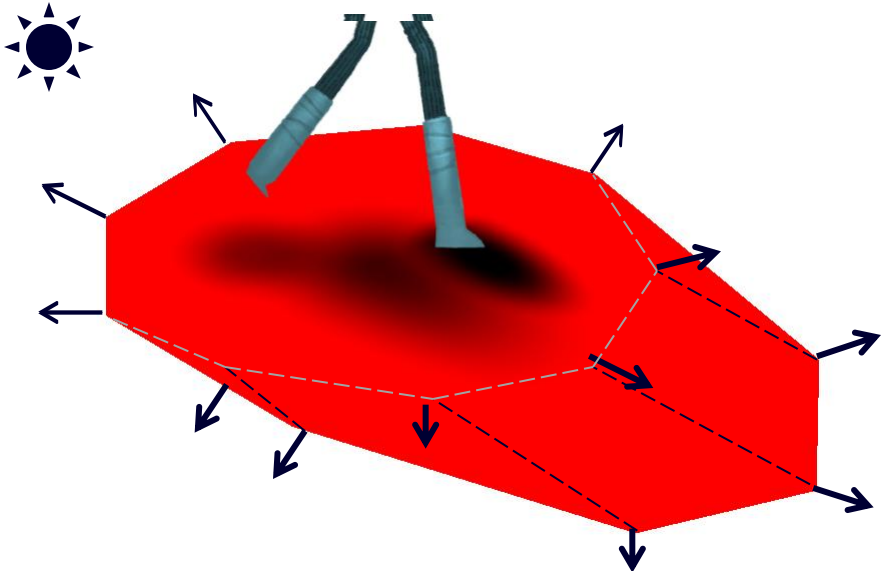
# Characters

- Lighting – Shadows
  - 3 shadow blobs (feet and body)
  - Distance to ground drives intensity and radius



# Characters

- Lighting – Shadows
  - Approximated directionality

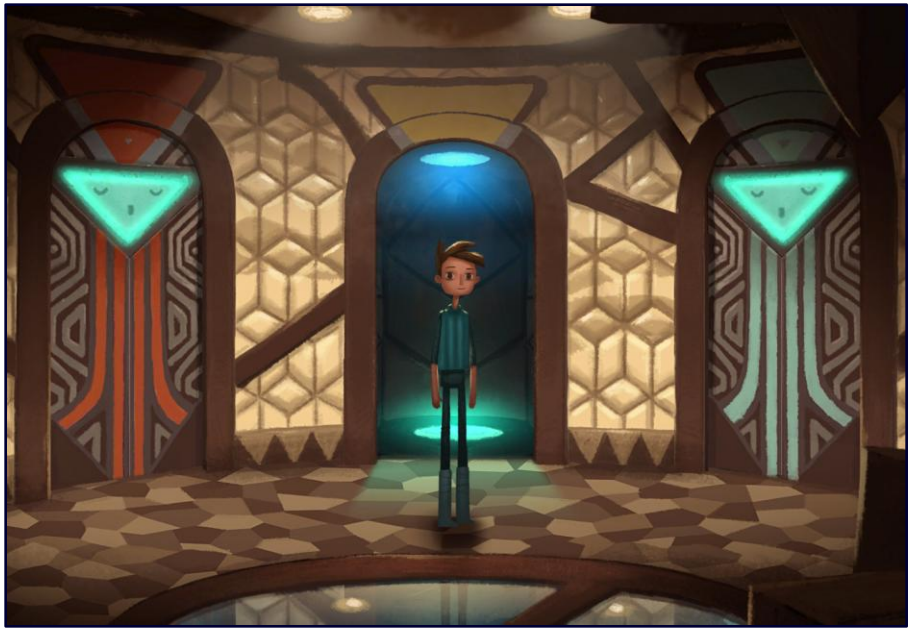


# Characters

- Lighting - Comparison


# Characters

- Lighting - Comparison



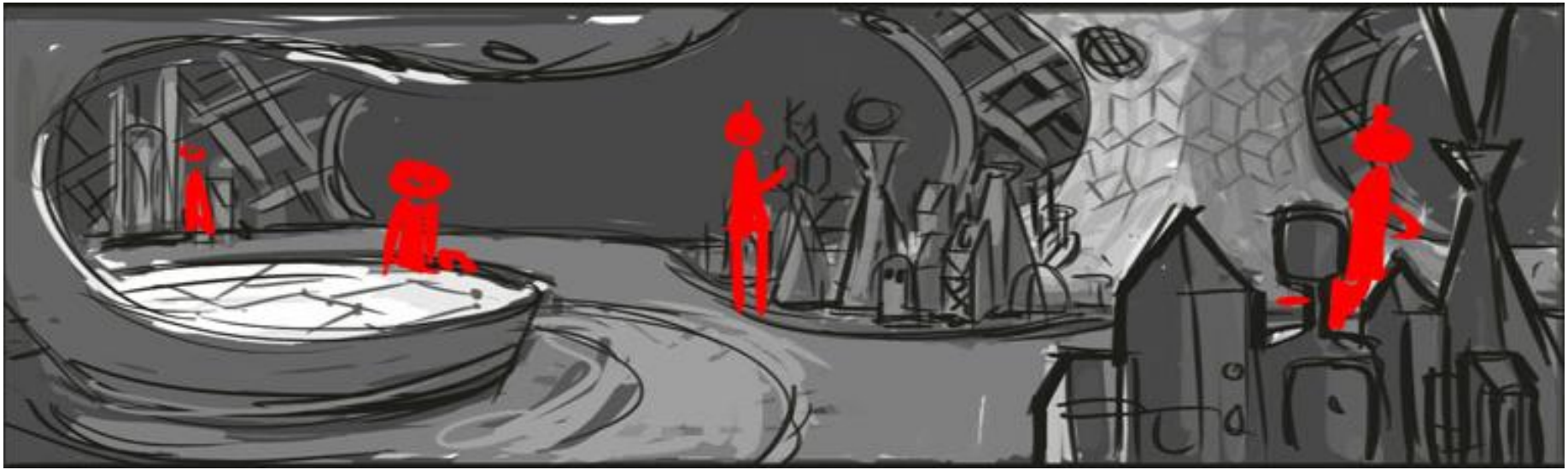
# Environments

- Goals
  - Painted in Photoshop
  - Support parallaxing
  - Multiple light states
  - Minimize memory overdraw

and

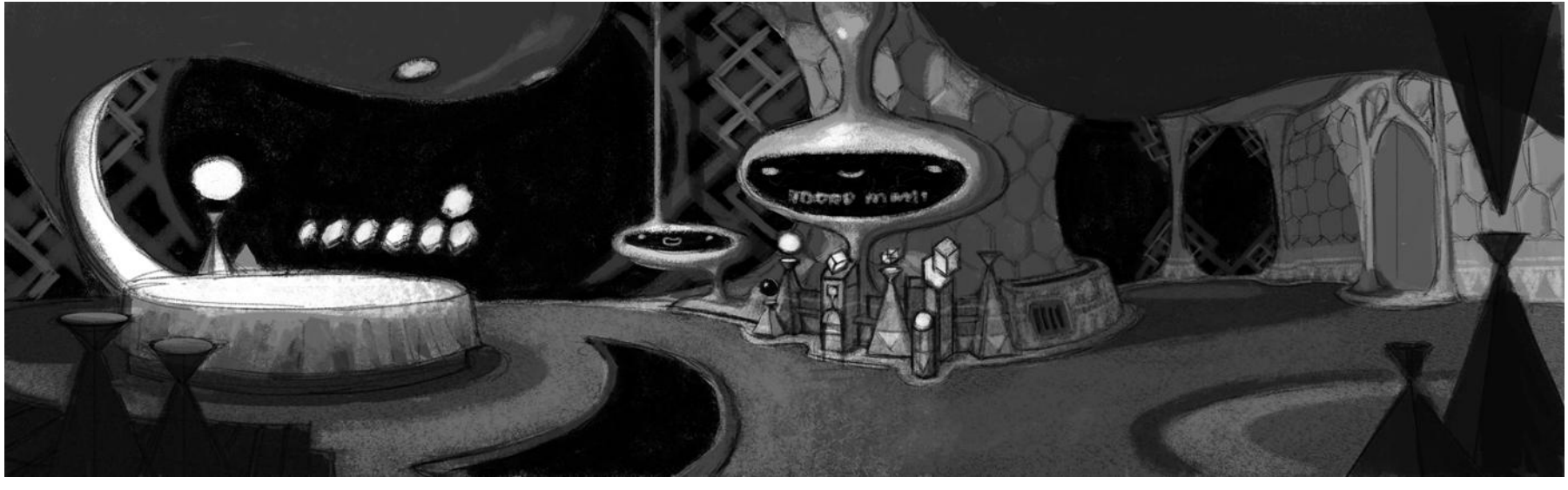
# Environments

- Authoring
  - Concept



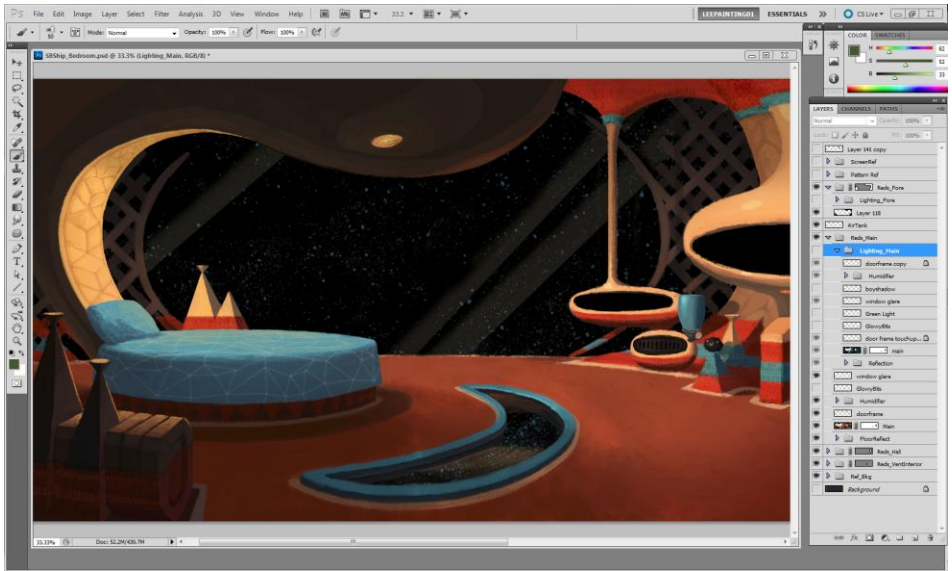
# Environments

- Authoring
  - Whitebox



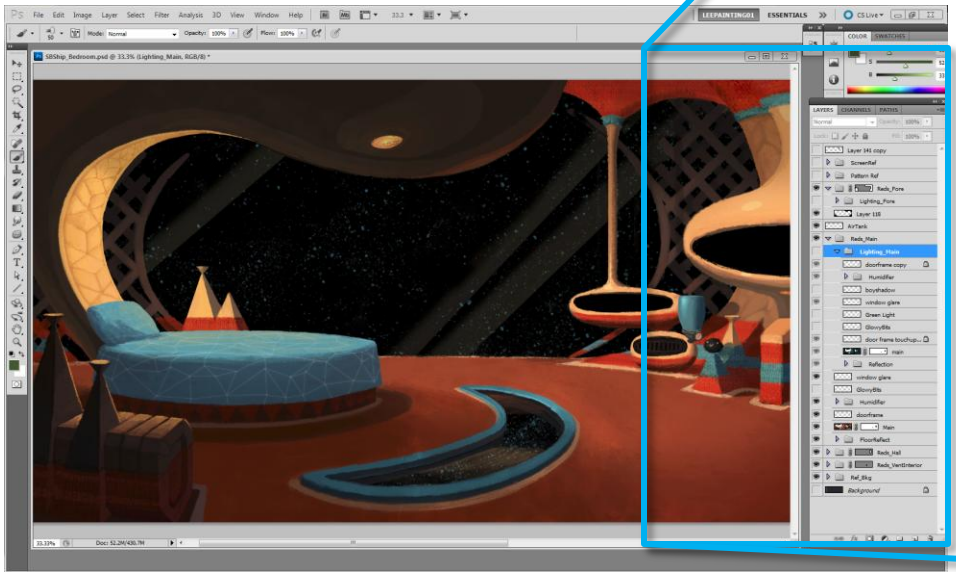
# Environments

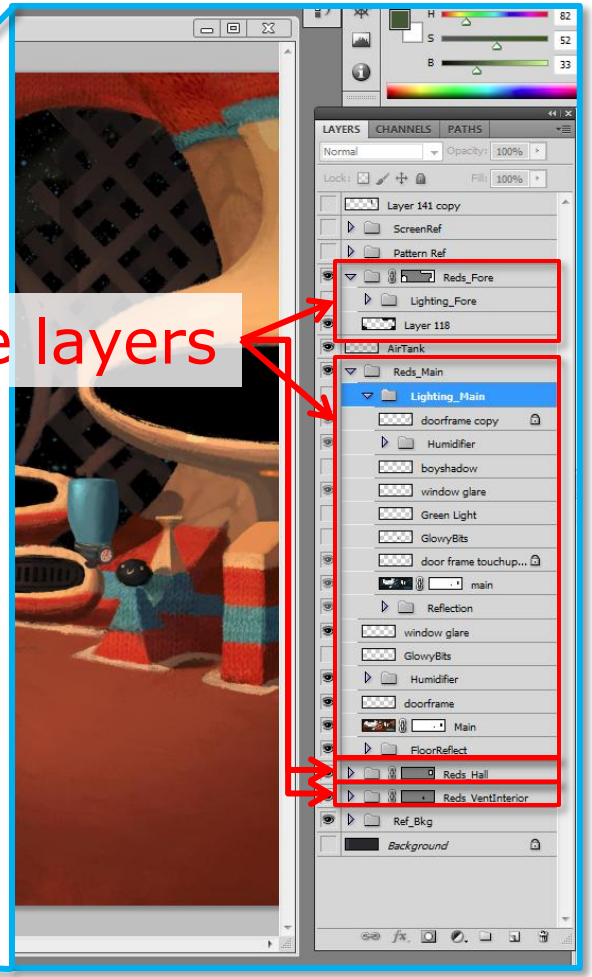
- Authoring
  - Final painting



# Environments

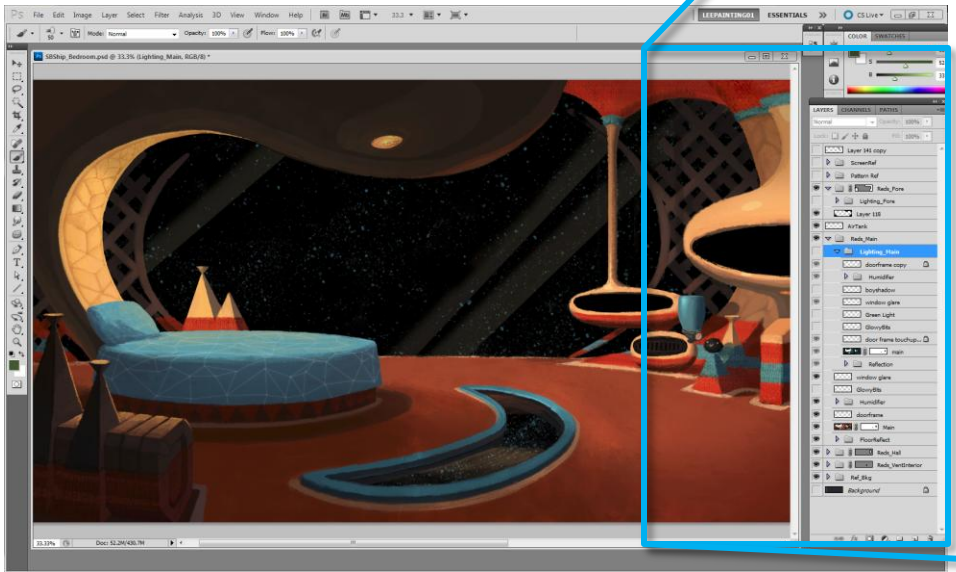
- Authoring
  - Groups become scene layers



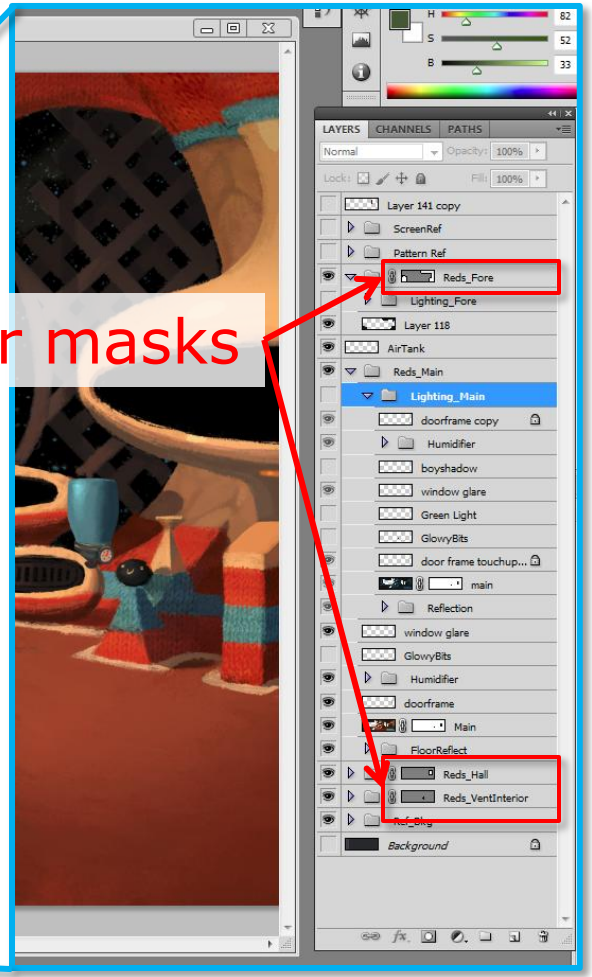


# Environments

- Authoring
  - Groups can have vector masks

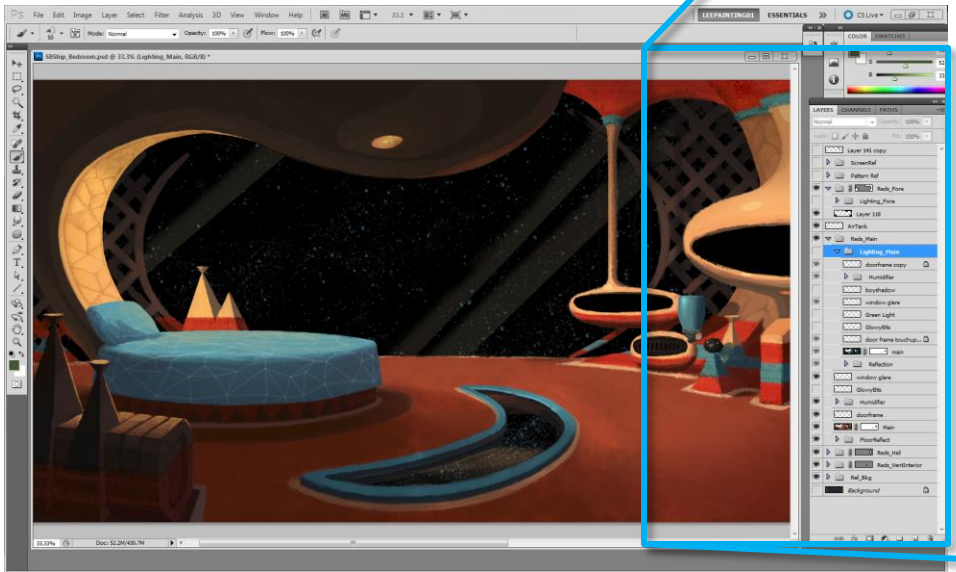


Vector masks

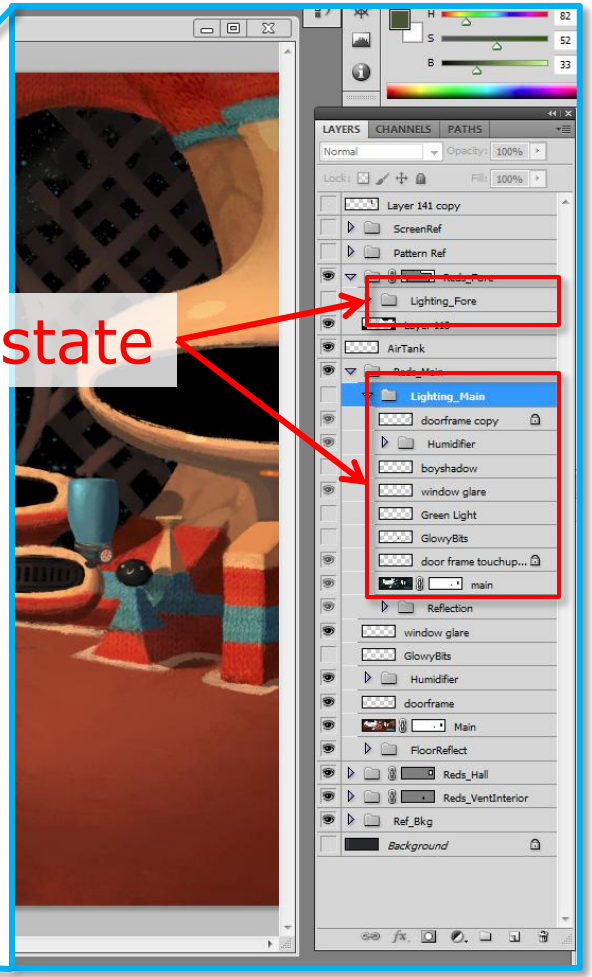


# Environments

- Authoring
  - Lighting groups define alternate state

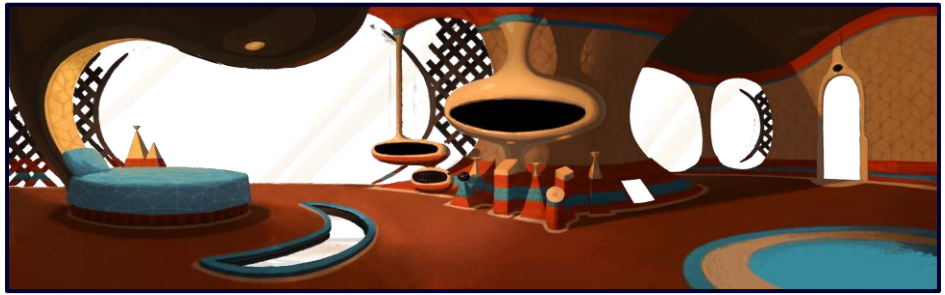


Light state

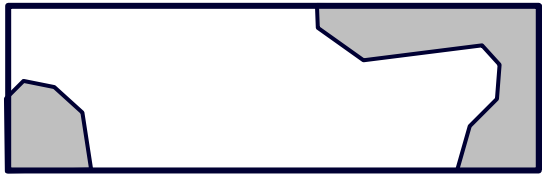


# Environments

- Authoring - Custom export script
  - Creates PNG and clip polygon per scene layer

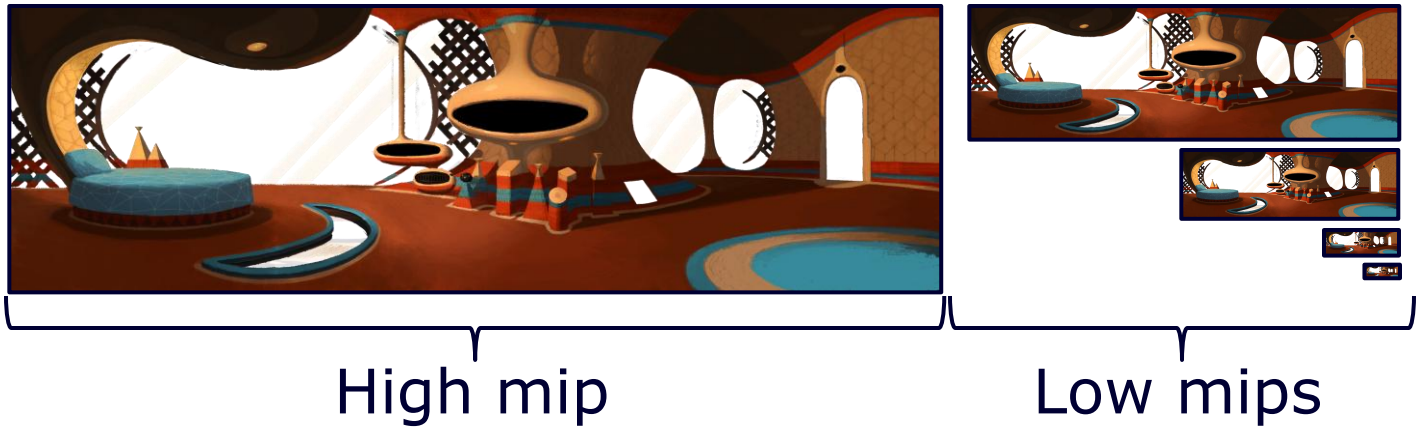


Vector masks



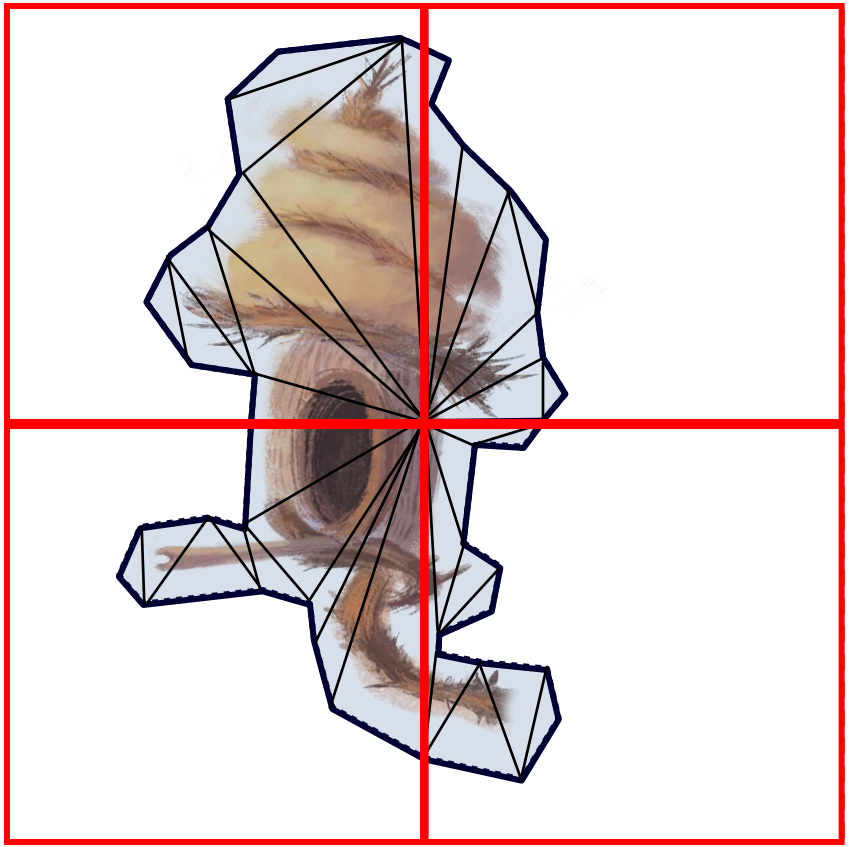
# Environments

- Data-build - Calculate mip-maps
  - Sharpen mips to counter loss of contrast
  - High and low mips are compressed separately
  - High mip only gets loaded on high-end platforms
    - minimize IO and memory footprint



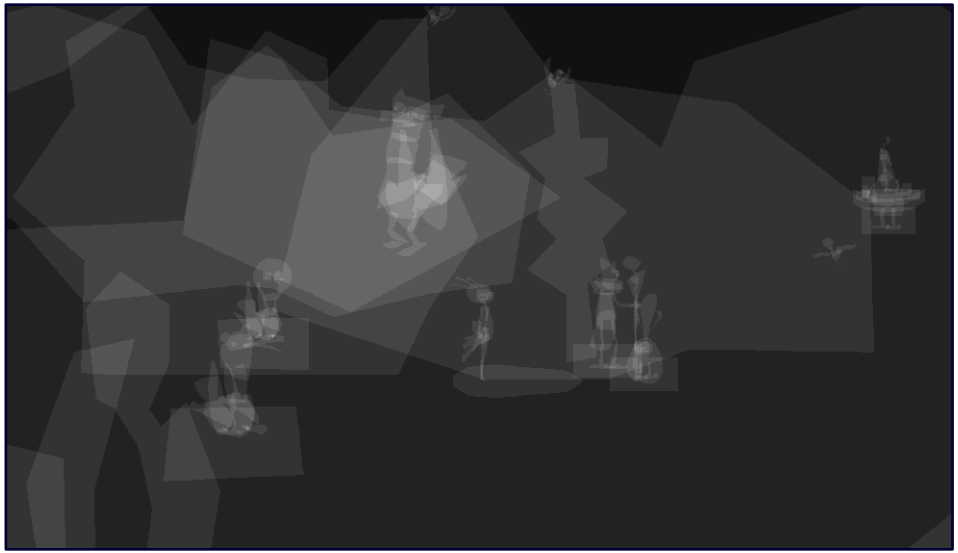
# Environments

- Data-build – Chunking
  - Split into GPU friendly textures
  - Calculate chunk polygons
    - Clipper library
  - Tesselate geometry
    - GLUtesselator
  - Omit empty chunks



# Environments

- Run-time
  - Clip masks minimize overdraw



# Environments

- Lighting
  - Blend between light states



# Shaders

- Goals
  - Optimized!
  - Permutations
    - No Übershaders!
    - Disable features on weak GPUs
  - Minimize impact of platform specific code
    - ETC1 needs extra alpha texture!
  - Support for #include and #if

# Shaders

- Optimization
  - No precompiled shaders in Open GL ES
  - Raw shader source is shipped
  - Drivers are very different
    - Amount and type of optimizations
    - Compilation speed
  - Extremely problematic

# Shaders

- Optimization
  - Offline GLSL optimization
  - Creates optimized shader based on original source
  - Maximizes performance
  - Minimizes size of shader source
  - Open source GLSL optimizer
    - Thank you Aras Pranckevičius!!!

# Shaders

- Optimization – Example (Original)

```
#include "common.fsh"

uniform MEDP vec4 g_vTintColor;
uniform MEDP float g_fCrossFadeIntensity;
uniform sampler2D g_samColor2;

varying MEDP vec2 uvVarying;

void main() {
    MEDP vec4 texColor = SampleMainTexture( uvVarying );

    if(g_bCrossFade)
    {
        MEDP vec4 texColor2 = SampleTexture( g_samColor2, uvVarying );
        texColor = mix(texColor, texColor2, saturate( g_fCrossFadeIntensity ));
    }

    gl_FragColor = texColor * g_vTintColor;
}
```

Include

Precision modifiers

Permutation

Helper functions

# Shaders

- Optimization – Example (Windows)

```
varying vec2 uvVarying;
uniform vec4 g_vTintColor;
uniform sampler2D g_samColor;
void main (){
   gl_FragColor = (texture2D (g_samColor, uvVarying) * g_vTintColor);
}
```

Perm: No cross fade

```
varying vec2 uvVarying;
uniform sampler2D g_samColor2;
uniform float g_fCrossFadeIntensity;
uniform vec4 g_vTintColor;
uniform sampler2D g_samColor;
void main (){
   gl_FragColor = (mix (texture2D (g_samColor, uvVarying),
      texture2D (g_samColor2, uvVarying),
      clamp (g_fCrossFadeIntensity, 0.000000, 1.00000)) * g_vTintColor);
}
```

Perm: Cross fade

# Shaders

- Optimization – Example (Android ETC1)

```
varying lowp vec2 uvVarying;
uniform lowp vec4 g_vTintColor;
uniform sampler2D g_samSplitAlpha;
uniform sampler2D g_samColor;
void main (){
    lowp vec4 tmpvar_1;
    tmpvar_1.xyz = texture2D (g_samColor, uvVarying).xyz;
    tmpvar_1.w = texture2D (g_samSplitAlpha, uvVarying).x;
    lowp vec4 tmpvar_2;
    tmpvar_2 = (tmpvar_1 * g_vTintColor);
    gl_FragColor = tmpvar_2;
}
```

Perm: No cross fade

# Shaders

- Optimization – Example (Android ETC1)

```
varying lowp vec2 uvVarying;
uniform sampler2D g_samColor2;
uniform lowp float g_fCrossFadeIntensity;
uniform lowp vec4 g_vTintColor;
uniform sampler2D g_samSplitAlpha;
uniform sampler2D g_samColor;
void main (){
    lowp vec4 tmpvar_1;
    tmpvar_1.xyz = texture2D (g_samColor, uvVarying).xyz;
    tmpvar_1.w = texture2D (g_samSplitAlpha, uvVarying).x;
    lowp vec4 tmpvar_2;
    tmpvar_2 = (mix (tmpvar_1, texture2D (g_samColor2, uvVarying),
         clamp (g_fCrossFadeIntensity, 0.000000, 1.00000)) * g_vTintColor);
    gl_FragColor = tmpvar_2;
}
```

Perm: Cross fade

# Shaders

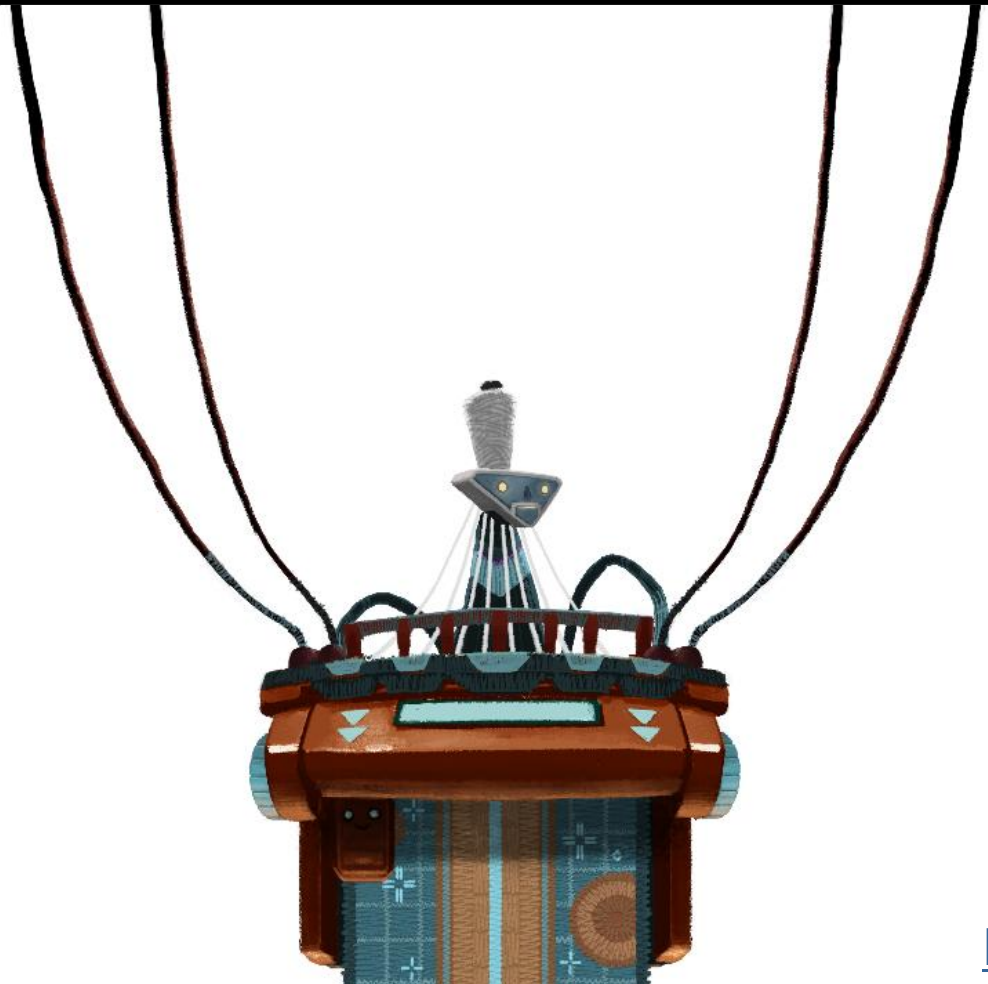
- Permutations – Data-build
  - Permutation variables
    - Flag (bool)
    - Enumeration
      - Drastically reduces the amount of permutations
      - n << n! + 2
  - Generate shader source for all permutations
  - ‘Select’ current state using C preprocessor of optimizer
  - Omit redundant shaders

# Shaders

- Permutations – Run-time
  - Only creates used permutation programs
    - ~15% are used (average)
  - Reuse vertex and fragment shaders
  - Permutation variable overrides for level-of-detail
  - Only one uniform state
    - Force re-apply uniforms when permutation changes

# Conclusion

- Think about platforms early on
  - Define the constraints
  - Artists are magicians!
  - Top-down and bottom-up scalability
  - Hide platform specific parts where possible



# Thank you!

Questions?

http://www.brokenagegame.com/

# References

- Open source libraries
  - GLSL optimizer
    - https://github.com/aras-p/glsl-optimizer
  - Clipper
    - http://www.angusj.com/delphi/clipper.php
- Further reading
  - Smedis: Bringing AAA graphics to mobile platforms
    - http://www.unrealengine.com/files/downloads/Smedberg_Niklas_Bringing_AAA_Graphics.pdf