



# Advanced Visual Effects in 2D Games

**Viktor Lidholt**

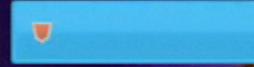
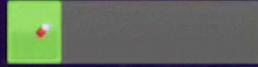
Lead developer – SpriteBuilder

GAME DEVELOPERS CONFERENCE®

MOSCONE CENTER · SAN FRANCISCO, CA

MARCH 2-6, 2015 · EXPO: MARCH 4-6, 2015

LEVEL: 2



BOMBS: 0

PAUSE





Shield and Energy Gauge

Energy Gauge

Score: 4666

Red button with number 3 and red sphere

Red sphere and button



# 2D visual effects

- Sprites
- Distorted polygons
- Particle systems
- Shader effects



# 2D visual effects

- Real-time graphics pipeline
- Effects using normal maps
- Color adjustments
- Blur & bloom
- Putting it all together



# Tools for building effects

- Creating assets
- Previewing effects
- Using good tools aids experimentation



2D lighting



Refraction



Reflection

Blur



Bloom

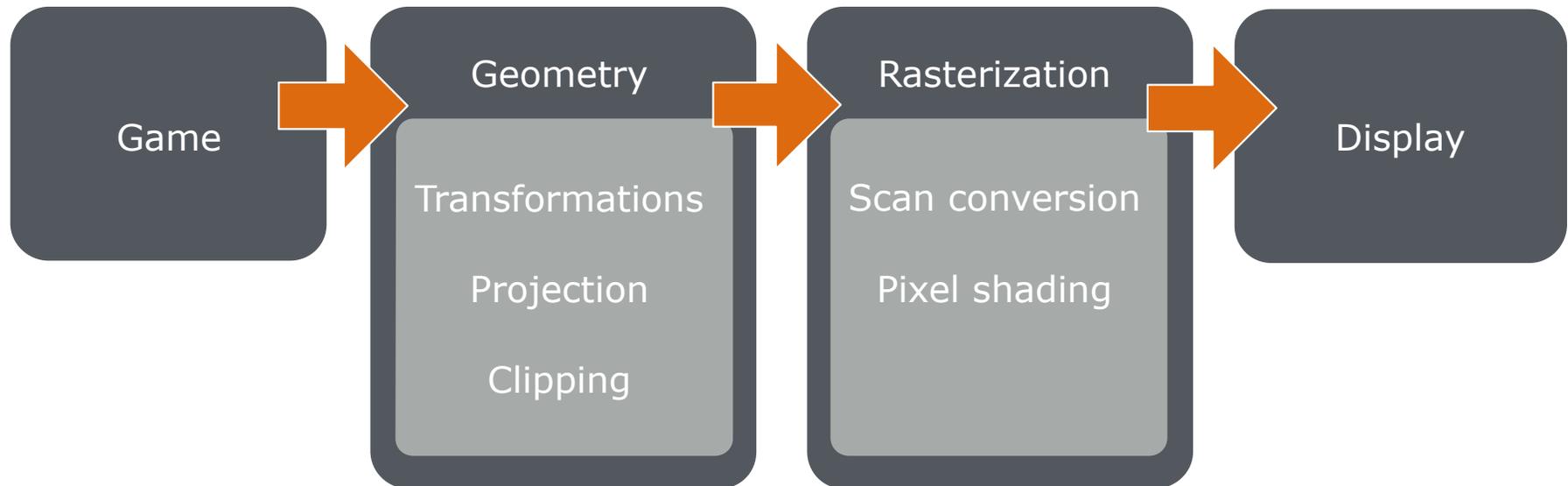




Color adjustments



# Graphics pipeline





# How effects are rendered

- Using sprites or polygons
  - Particle systems
  - Motion streak effects
- Using shaders on the graphics card
  - Pretty much any effect



# What is a shader?

- Programs compiled on the graphics card
- Platform dependent
  - GLSL / OpenGL
  - Metal shading language
  - HLSL / DirectX
- C or C++ like language



# What is a shader?

- Introduced in OpenGL ES 2
- Not supported on older hardware
- Performance varies with graphics card
- Requires testing on a multitude of devices



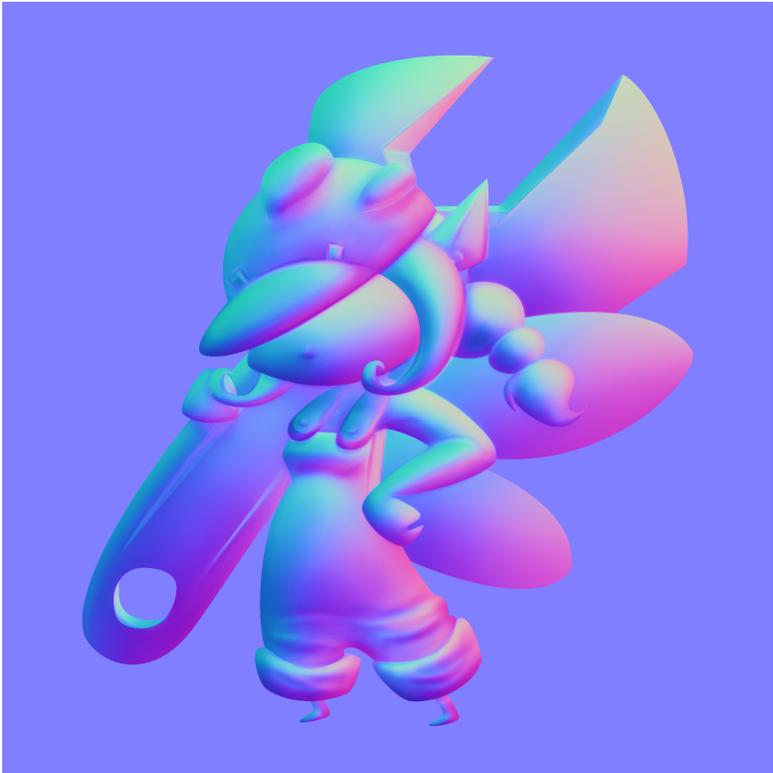
# What is a shader?

- Vertex shaders
  - Modifies the location of vertices
- Pixel (fragment) shaders
  - Computes the color of each rendered pixel
  - Can sample the environment
  - Can use multiple textures



# Normal maps

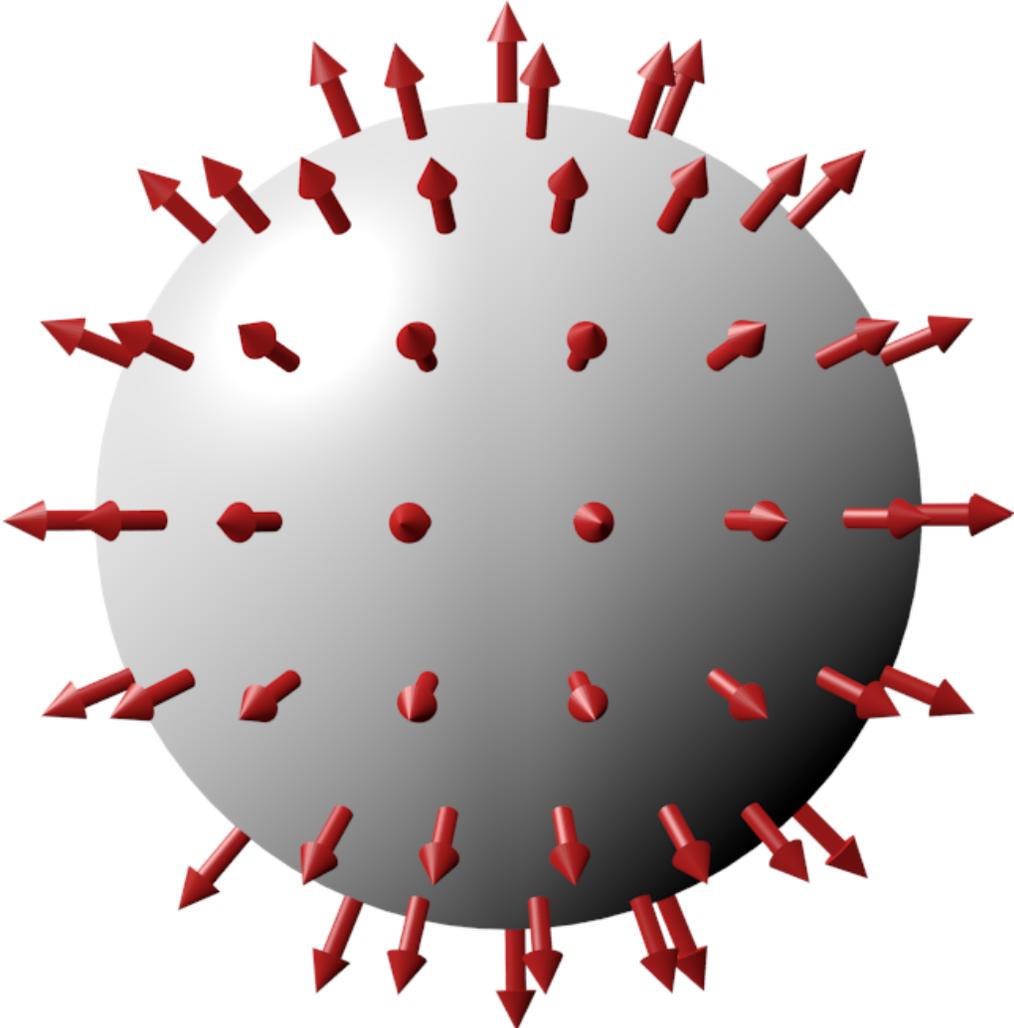
“A normal map is an image where the RGB components correspond to the X, Y, and Z coordinates, respectively, of the surface normal.”





# What is a normal?

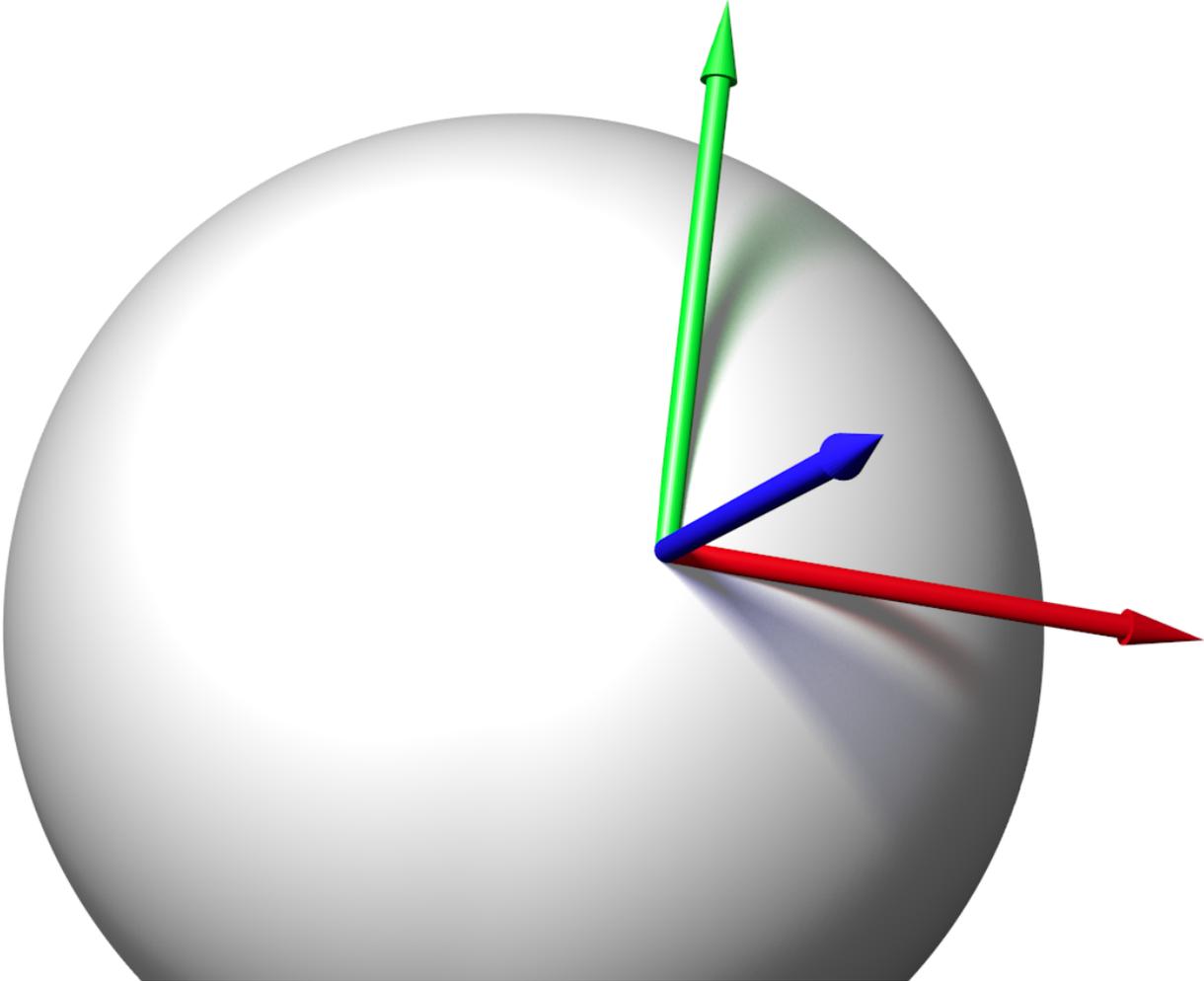
A normal is a vector that is perpendicular to a given object's surface





# What is a normal?

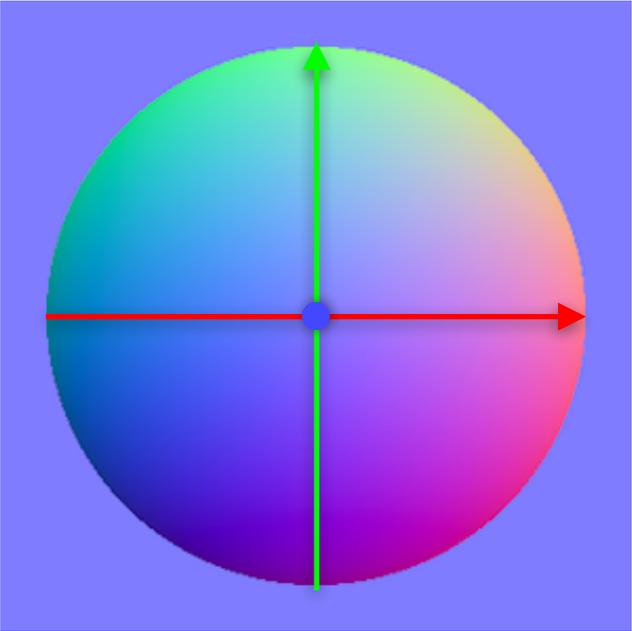
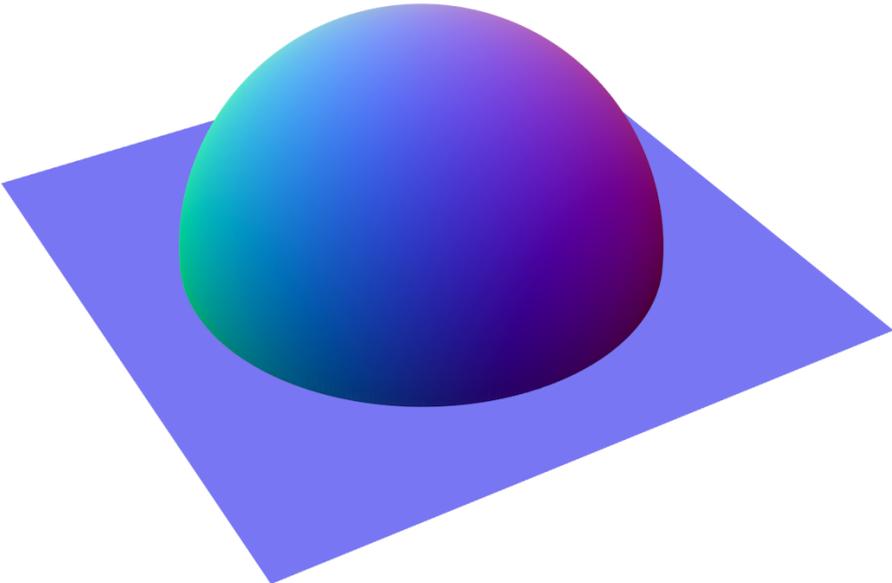
- Any vector in 3D space can be decomposed into three components  $(x, y, z)$
- These components can be mapped to the RGB values in an image





# Normal maps

- A normal map depicts the normals of a 3D object as viewed from a specific direction
- Each pixel's color corresponds to the surface normal at that position



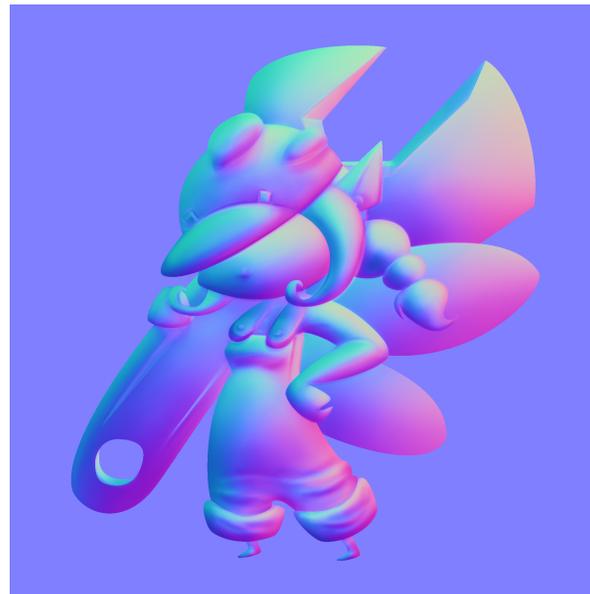


# 2D lighting

- Fast to render on any modern hardware
- Relatively easy to implement
- The real problem is the complexity of building the normal maps



Diffuse map



Normal map



Lighted sprite



Shield and Energy Meter

Energy Meter

Score: 4666

Red button with number 3



# Tweaking lighting

- Ambient light
  - Base lighting
- Specular light
  - Highlights or gloss
  - Color & intensity can be modified
  - Custom specular texture maps



Plain

Ambient

Specular



# Making it look great

- Requires experimentation
- Using tools make the process faster
- Find or make the right tools for you



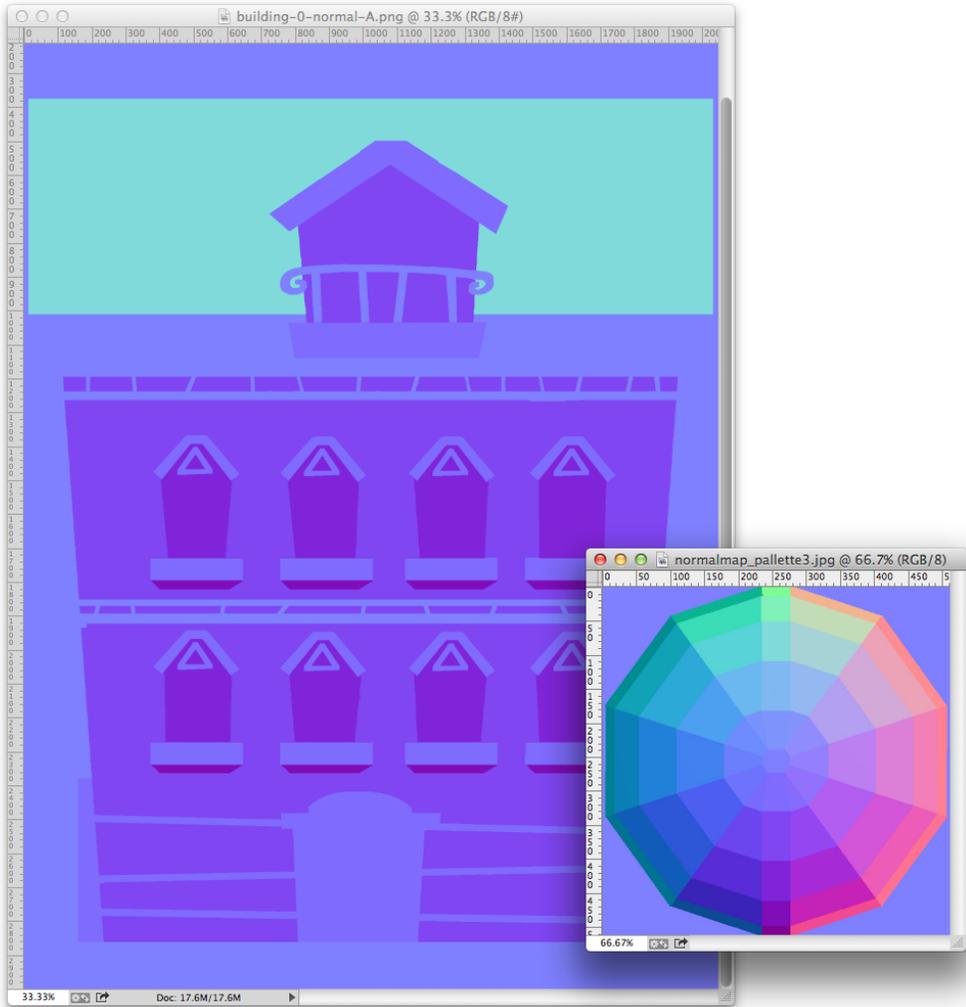
# Creating normal maps

- Manually drawn
- Generate in 3D application
- Build from height map
- Specific tools
- Blending normal maps



# Manually drawn

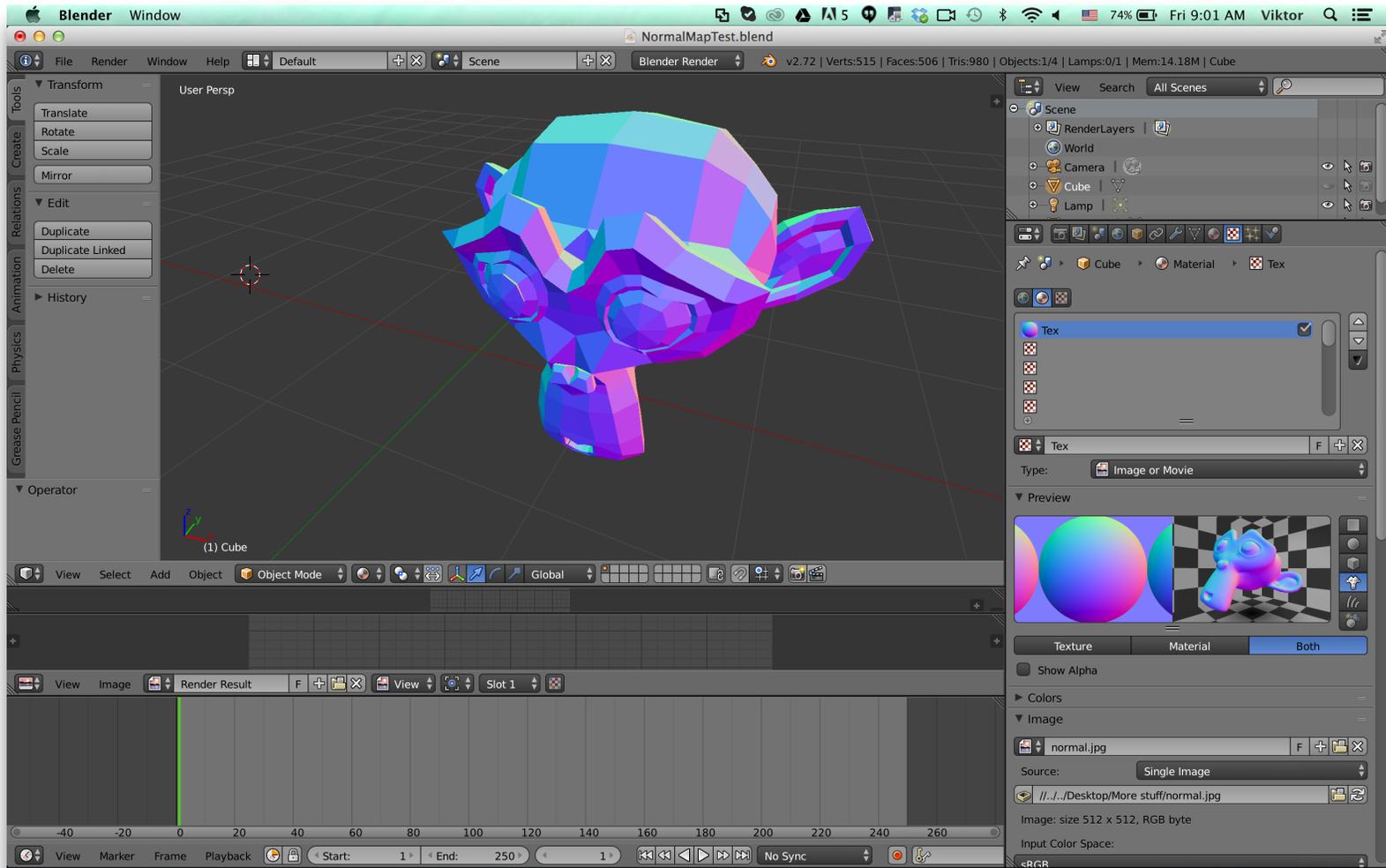
- Use the normal sphere to pick colors
- Good for sharp edges
- Anti-alias can have unexpected effects
  - Be careful while drawing
  - Be careful while scaling down





# Generate in 3D application

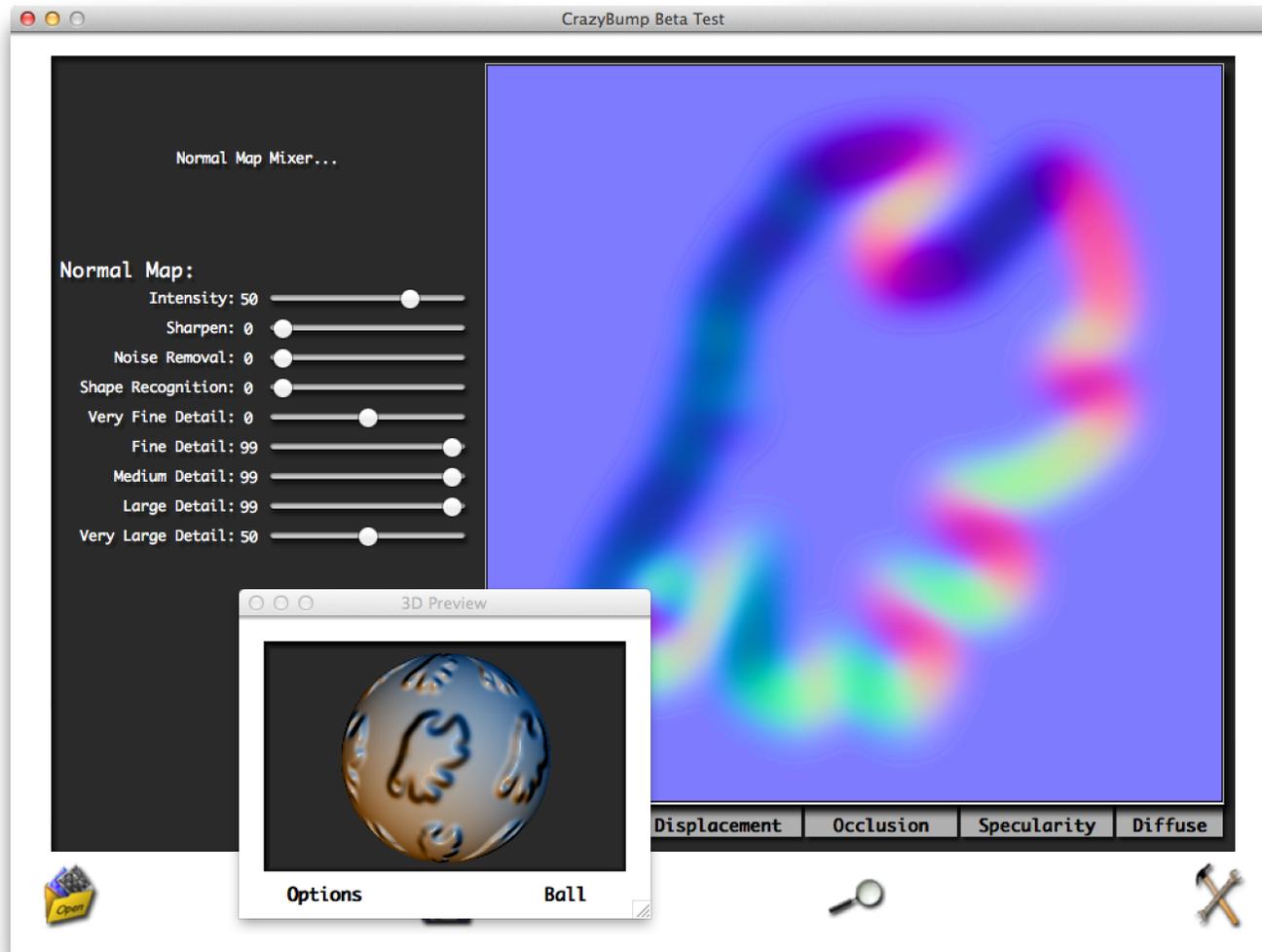
- Good, predictable results
- Easy to setup in 3D program
- Requires 3D models of your art

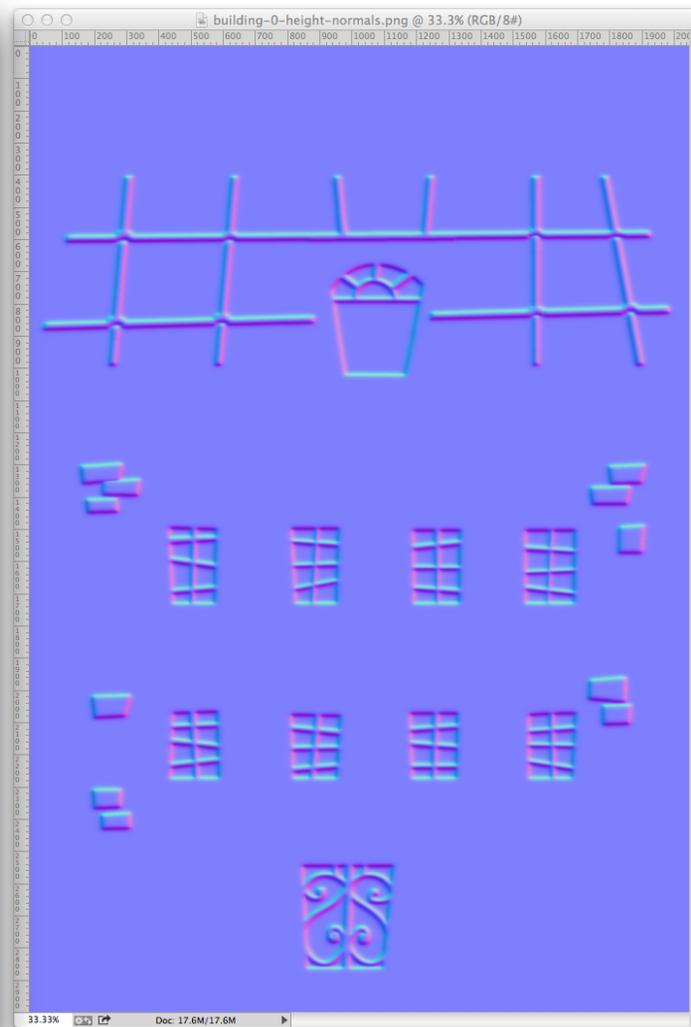
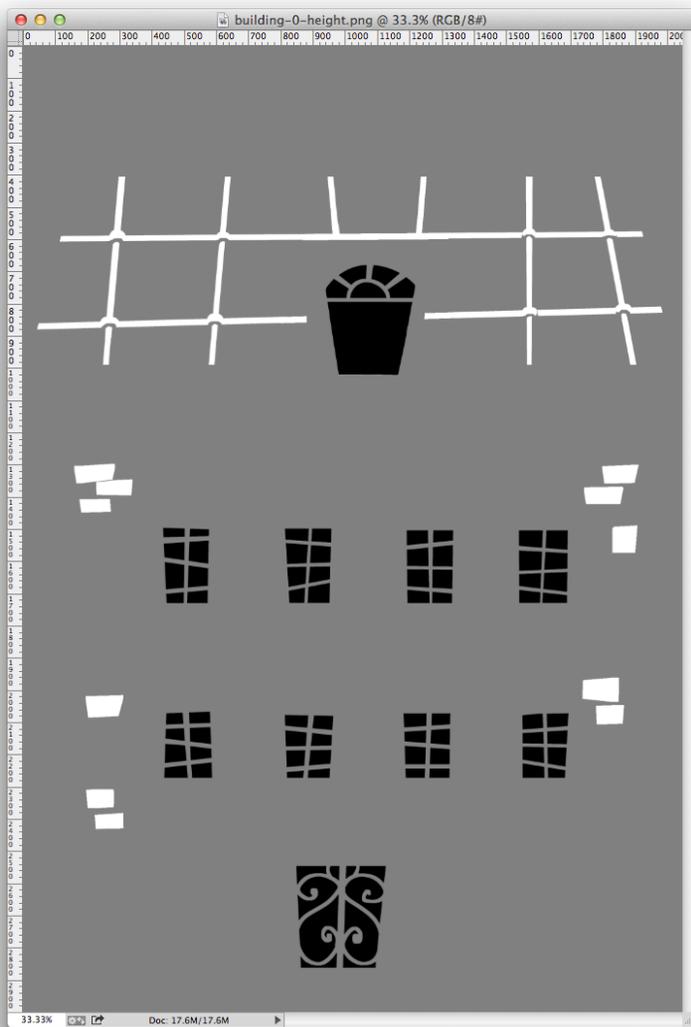




# Build from height map

- Height maps can be manually drawn
- Good for details
- Results may need tweaking

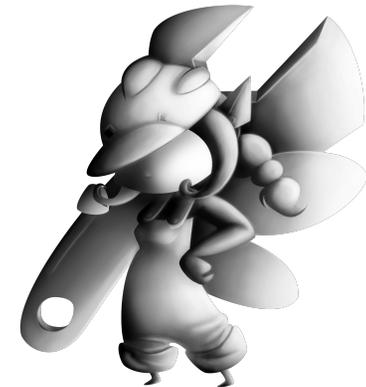
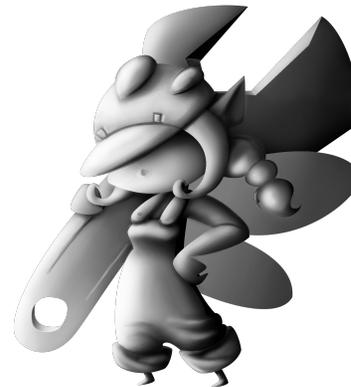
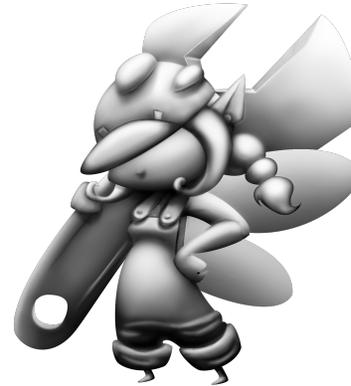
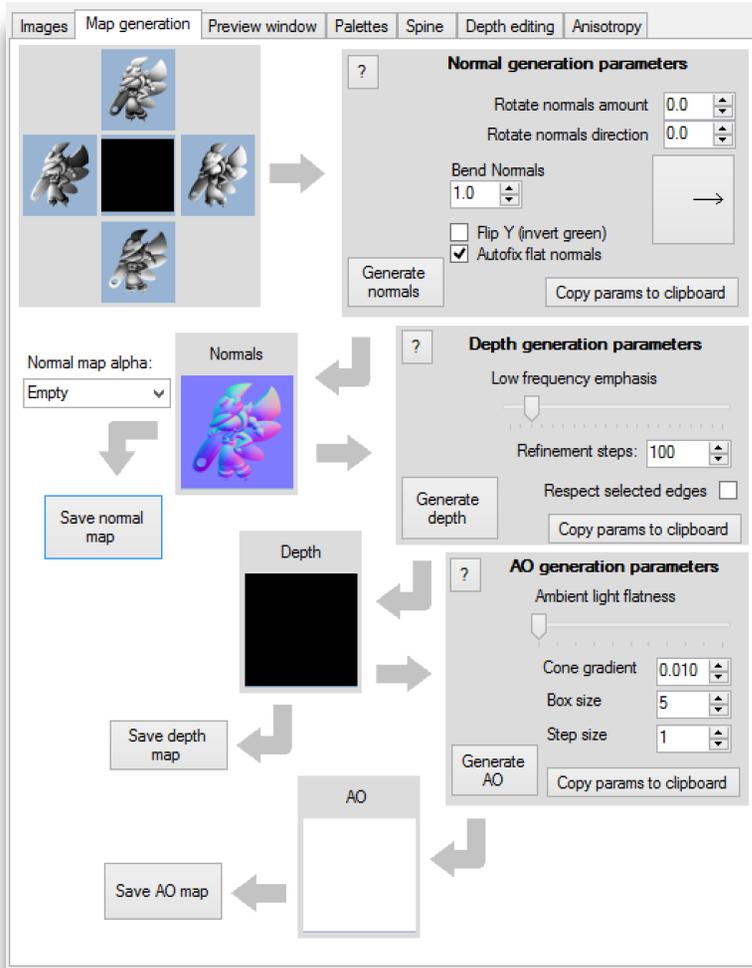


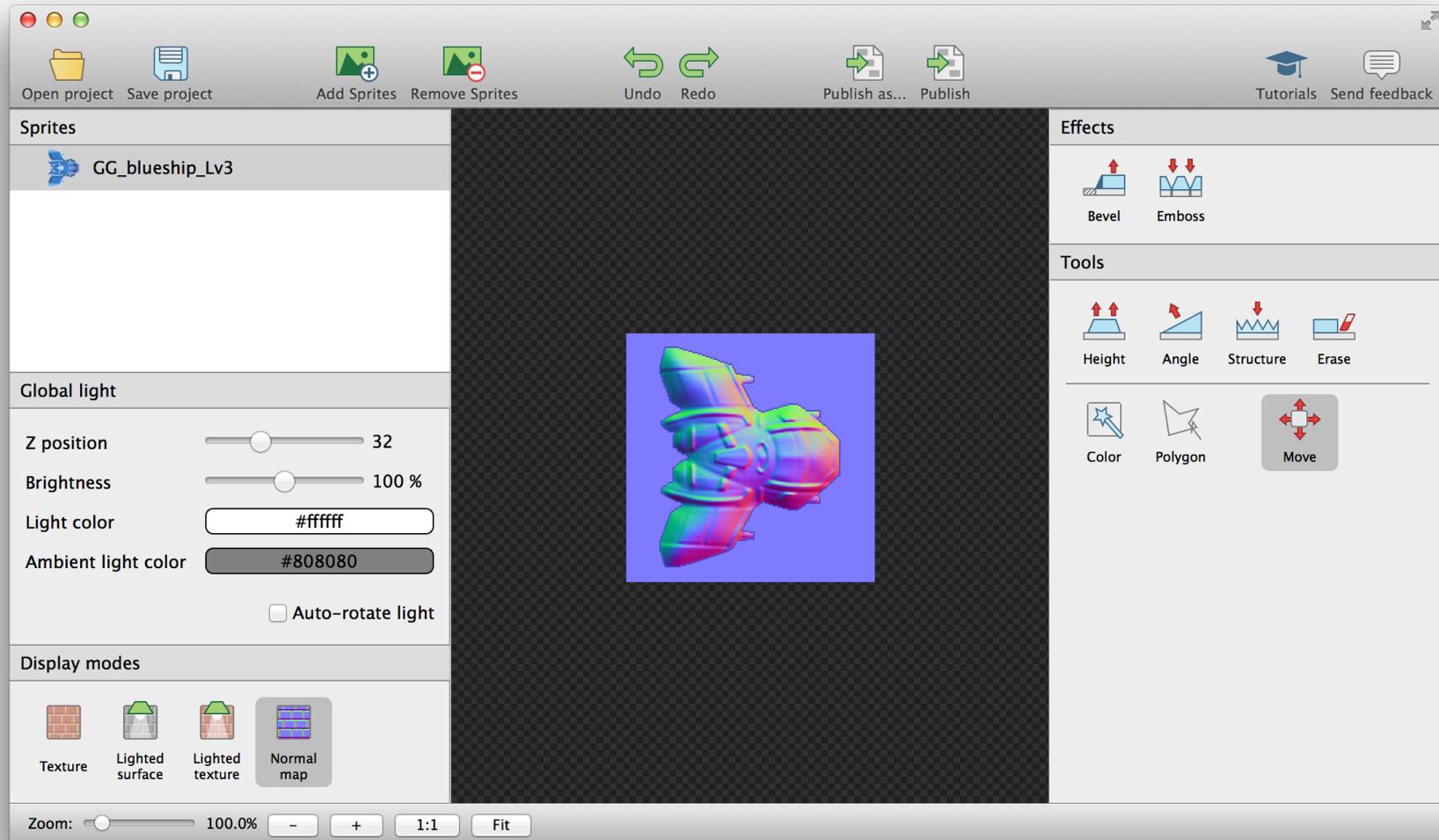




# Specific tools

- SpriteLamp
  - Great results for organic shapes
  - Requires a lot of extra art
- SpriteIlluminator
  - Drawing program for normal maps
  - Some sprites can be tricky

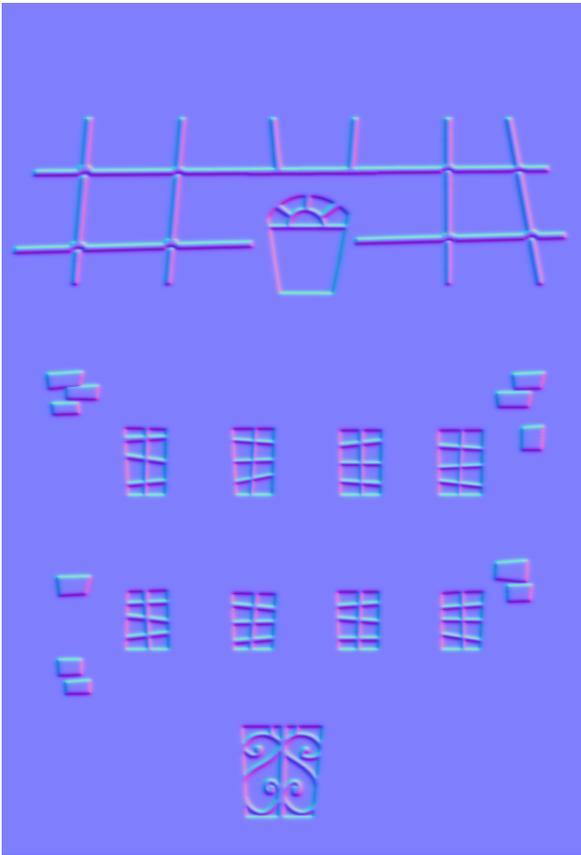






# Blending normal maps

- Simply blending / adding colors won't give expected results
- Special purpose tools available

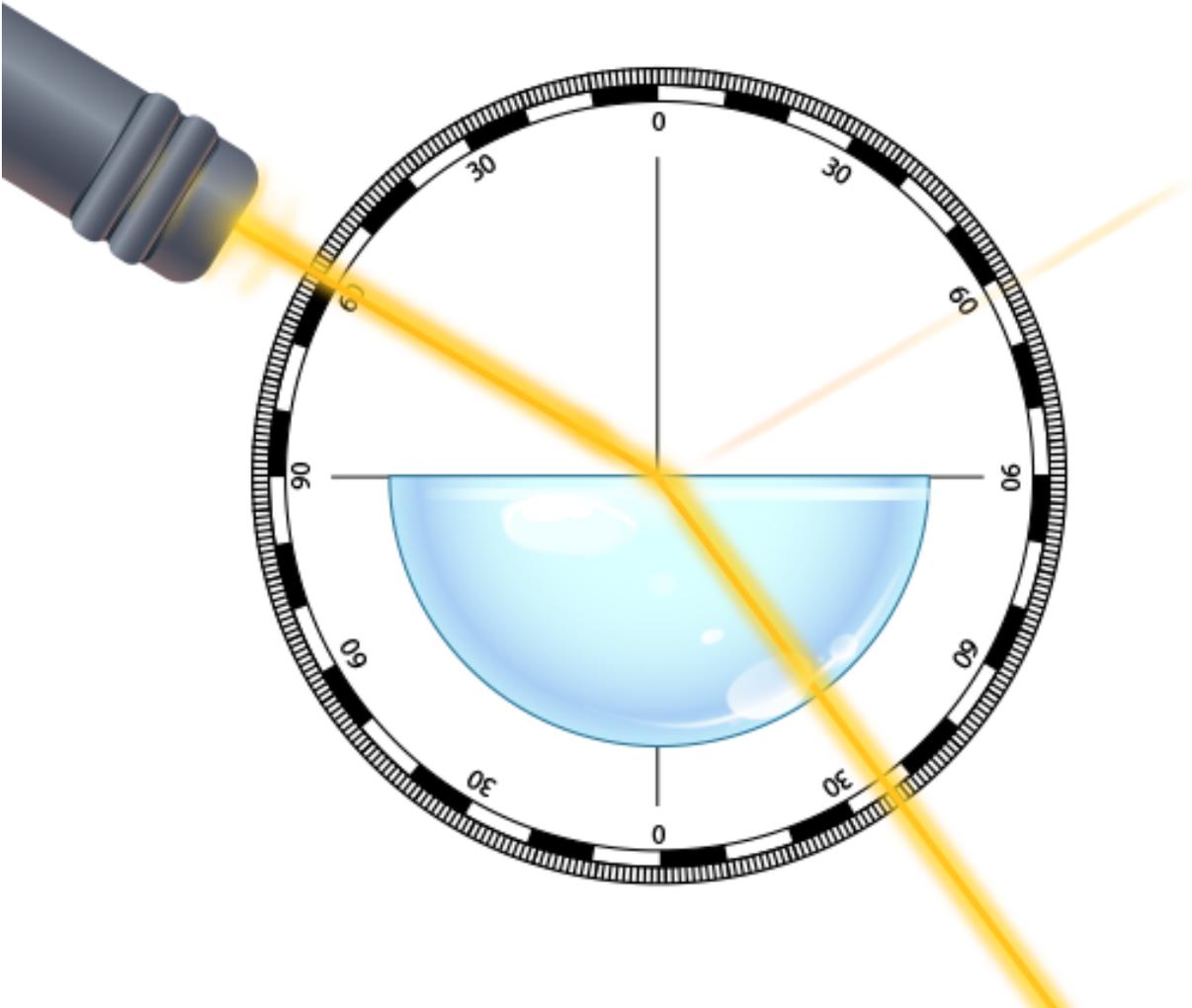






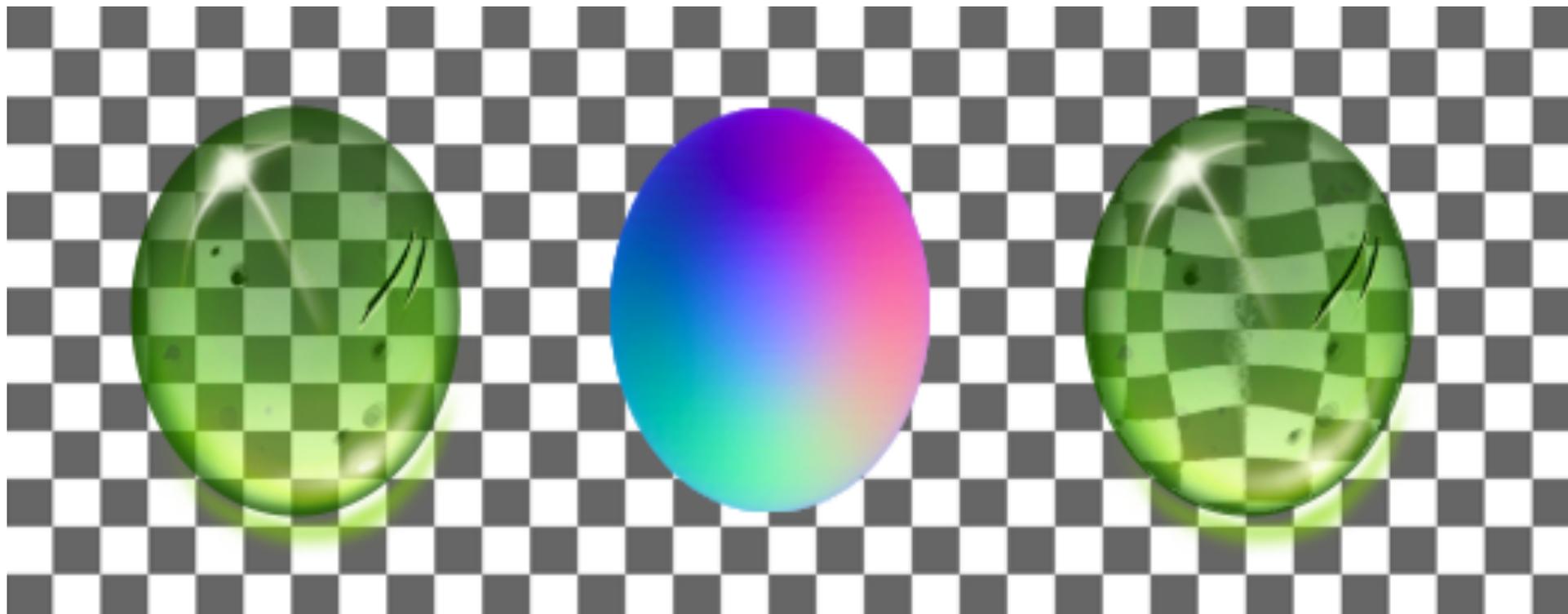
# Refraction effects

- Simulates light breaking through transparent materials
- Uses normal maps in combination with environment (refraction) maps





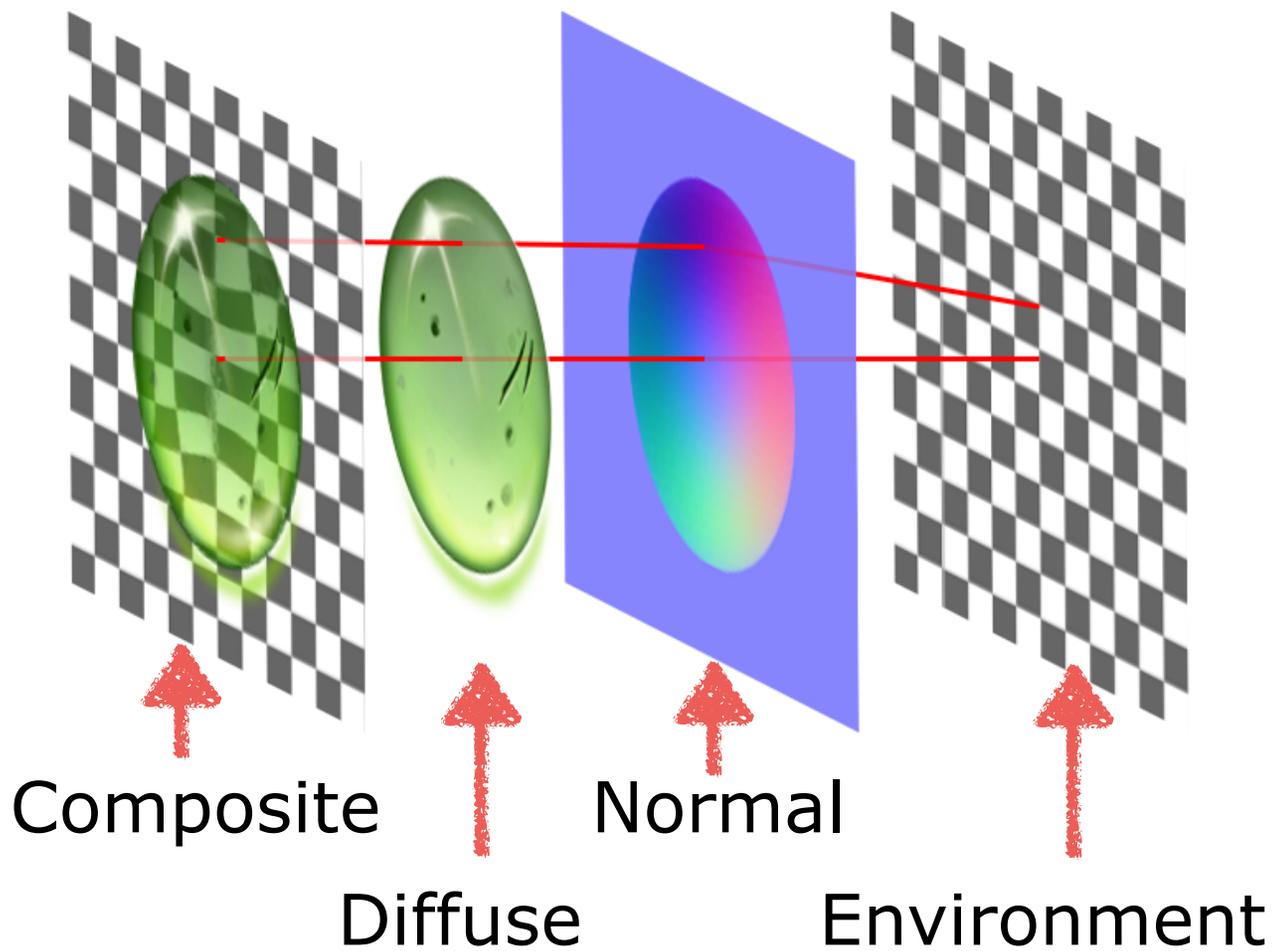
Refraction



Diffuse map

Normal map

Refraction





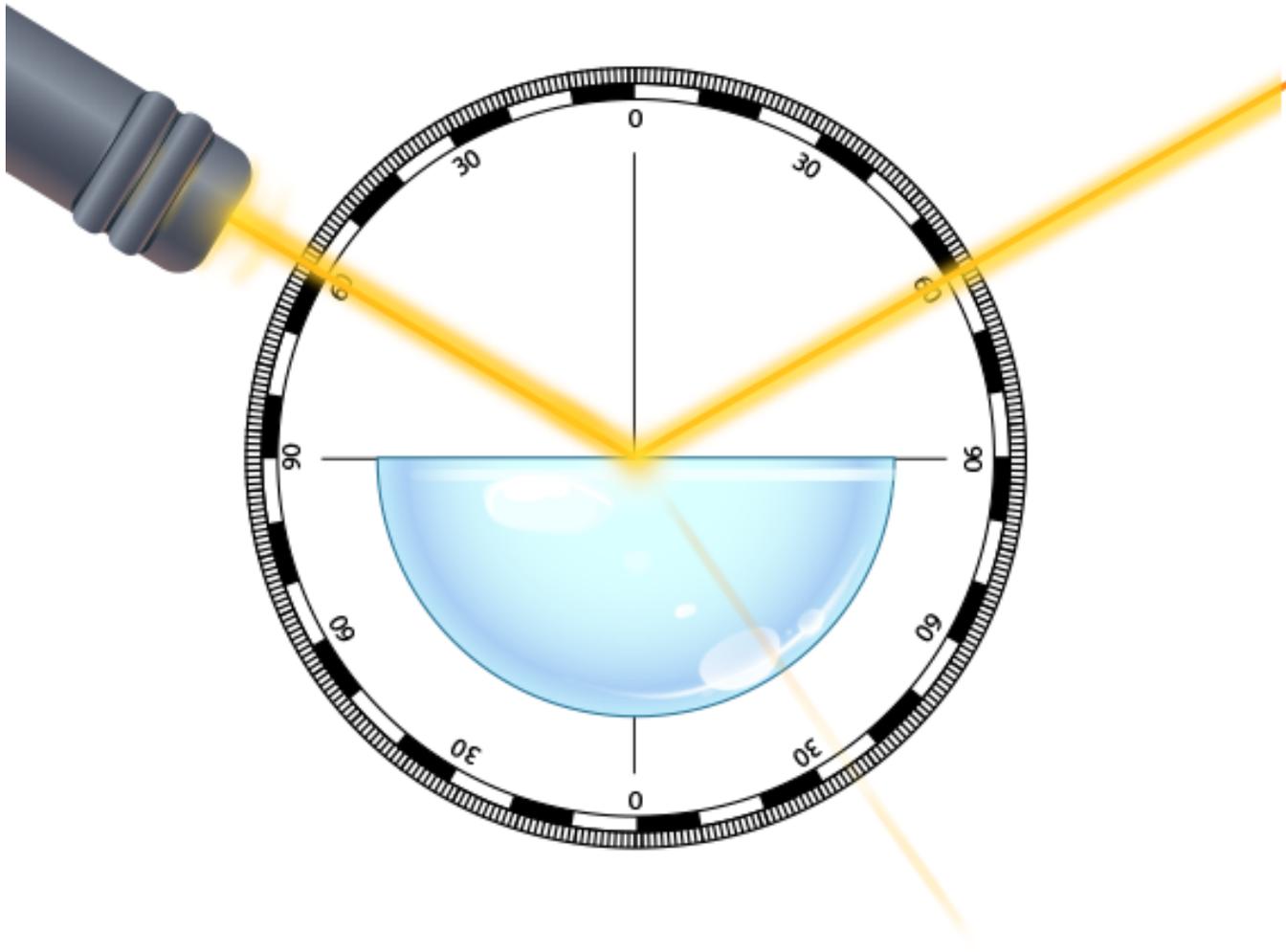
# Environment map

- Single sprite
  - Fast
  - Cannot render intermediate objects
- Render texture
  - Slower
  - Can have animations in the environment



# Reflection effect

- Much like refraction, but simulates reflection
- Uses normal maps in combination with environment (reflection) maps





Reflection

13

.017

20 - 0



Reflection



# Tips on normal map effects

- Combine refraction and reflection to create glass-like effects
- Environment maps can be moved or rotated to appear more dynamic
- Effects doesn't need to be perfect to look good



# Color adjustments

- Reduce number of assets
- Highlight game elements
- Improve animations
- Transitions



# Color adjustments

- Saturation
- Contrast
- Brightness
- Hue



Saturation



Contrast



Brightness

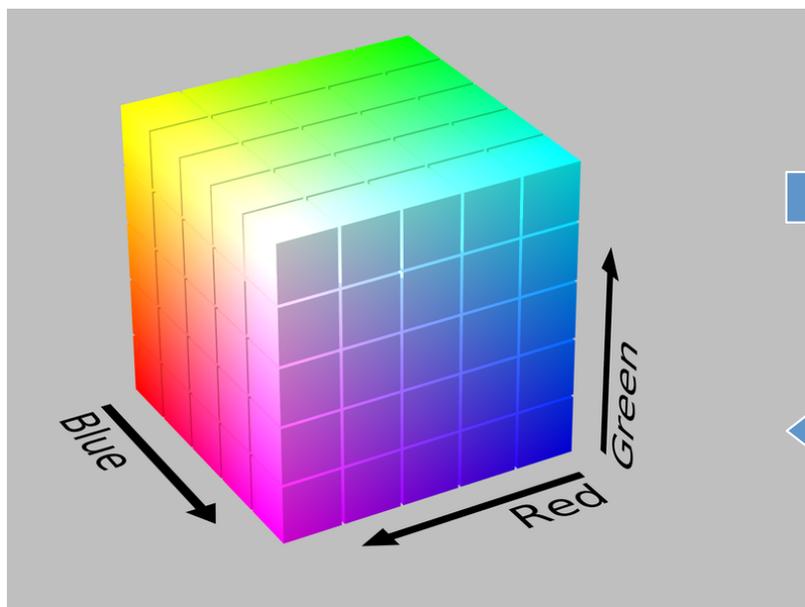


Hue

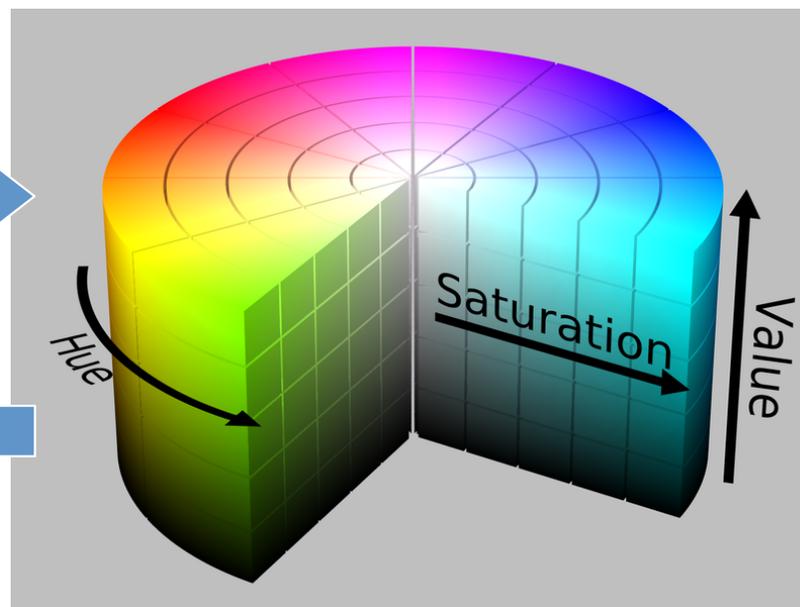


# Color adjustments

- Saturation, Contrast & Brightness can all be done in the RGB color space
- Shifting the hue requires conversions between color spaces



RGB



HSV



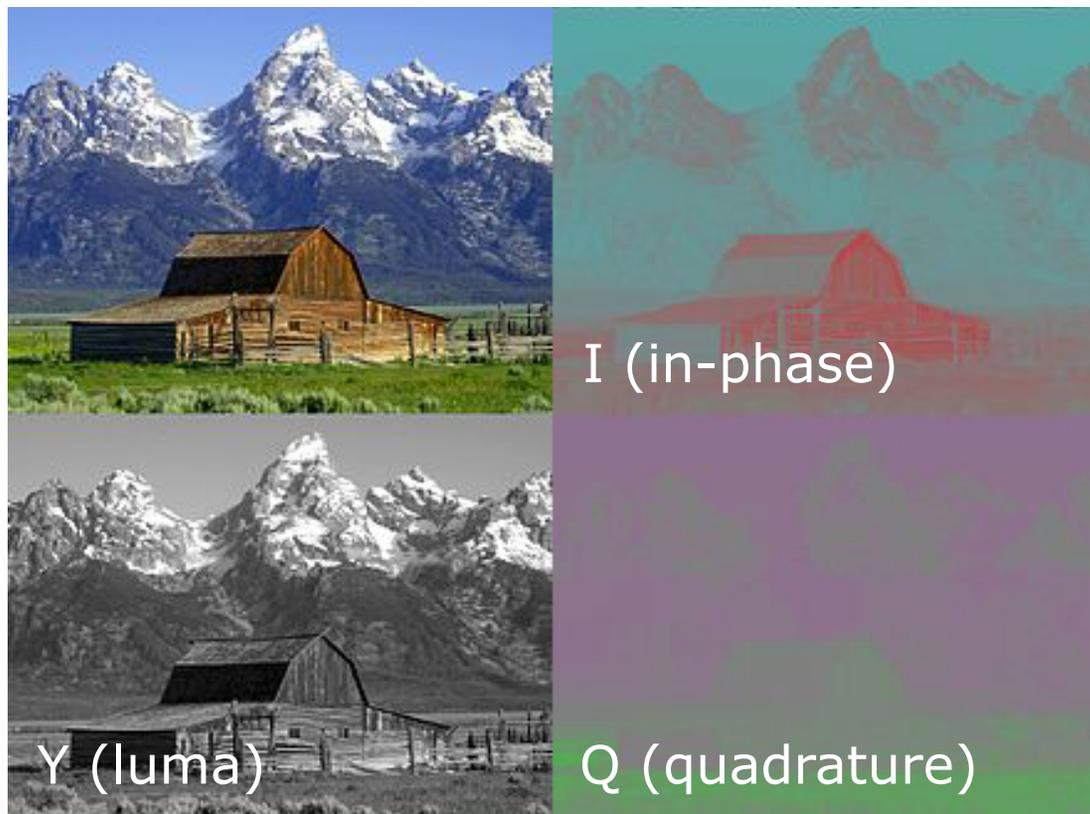
# Shifting hue

- Converting to HSV is slow
- Hue shift can be approximated in YIQ color space
- Shifting hue using YIQ is twice as fast, but not perfect

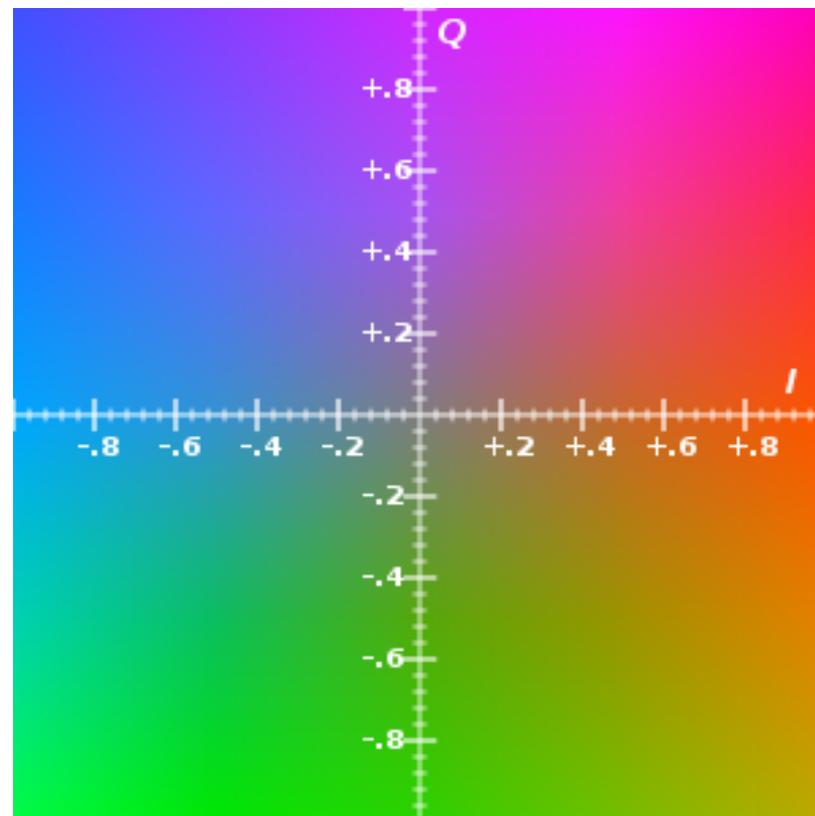


# YIQ colorspace

- Used by NTSC color TV system
- Luma, in-phase & quadrature
- Conversion to/from RGB is done by a simple matrix multiplication
- Can shift the hue by rotating around the luma axis



YIQ channels



Color cube at luma = 0.5



HSV

Preserves saturation



YIQ

Preserves luminance



# Reducing number of assets

- Color sprites or part of sprites, instead of adding more textures saves memory
- Can impact performance
  - Color adjustments are slower to render than plain sprites
  - Adding different shaders break sprite batching





# Blur & bloom

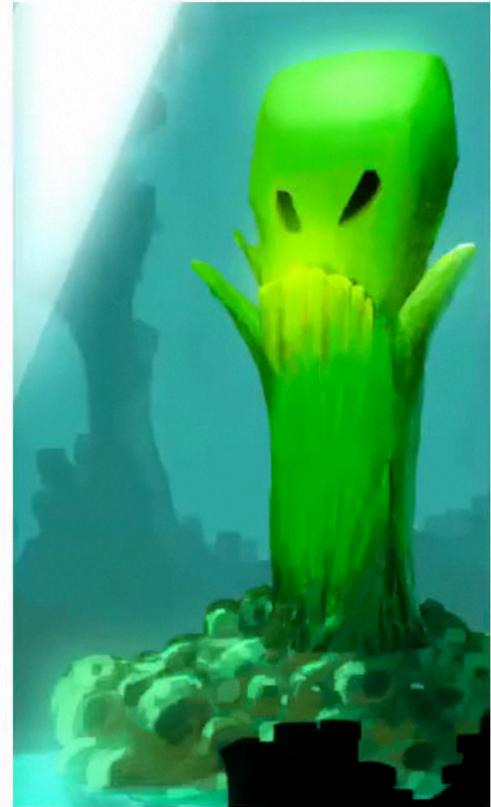
- Blur
- Bloom
- Drop shadow
- Glow



Original



Blur

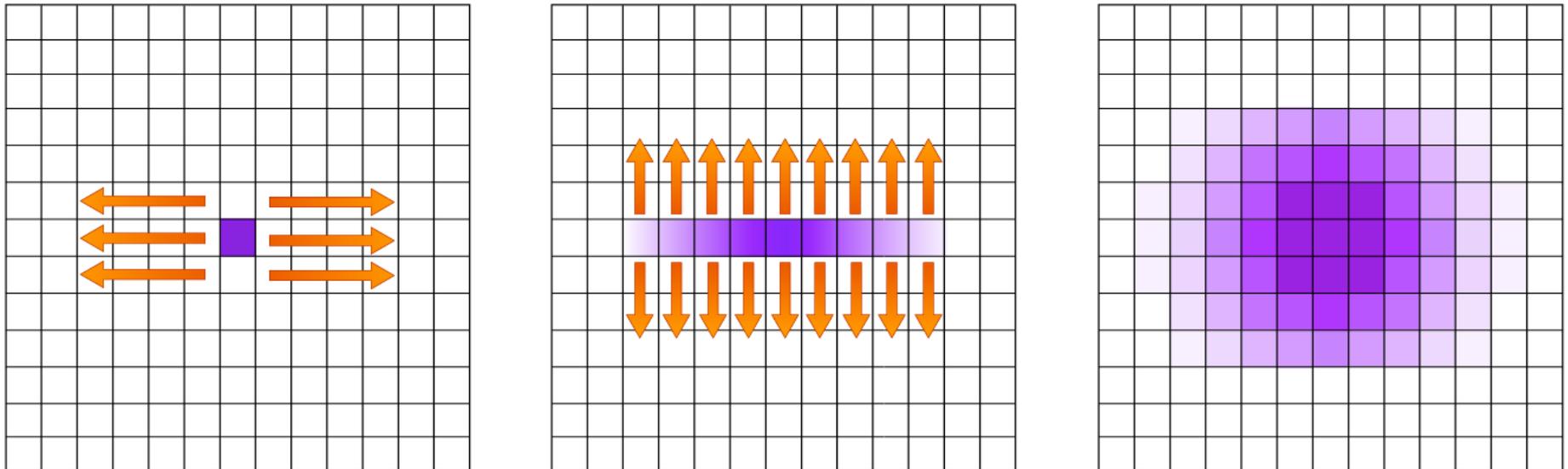


Bloom



# Blur

- Doing correct blurs are very expensive
- Can be simplified with multiple render passes
- Different implementations are faster on different devices



Blur using multiple render passes



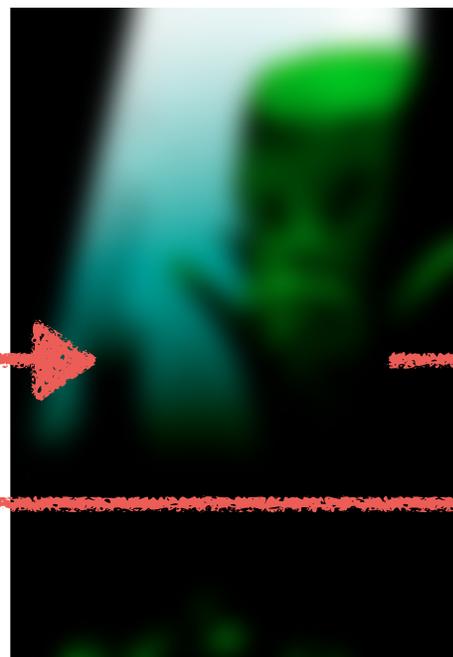
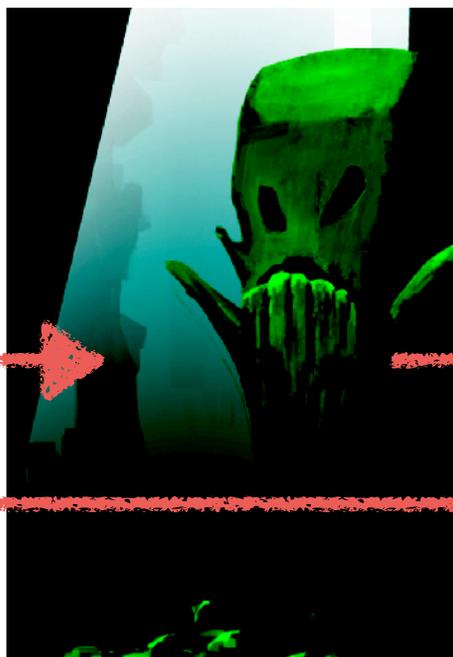
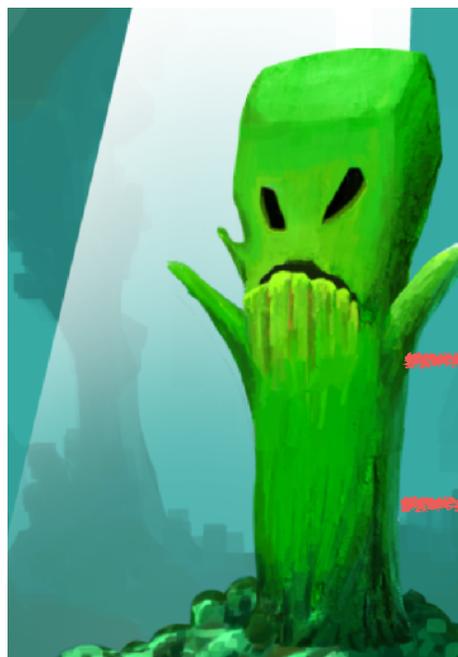
# Blur optimizations

- Use pre-rendered images
  - Dissolve/blend between can simulate dynamic results
- Buffer the results in a render texture
  - Useful when the blurred areas are not dynamic



# Bloom

- Feathers of light extending from brightly lit areas
- Created by combining blur with filtering
- Expensive to render



Original

Filtered

Blurred

Added



# Putting it all together

- Tweaking performance
- Combining effects



# Good toolsets for effects

- Allows visual exploration
- Makes it easier to tweak performance
- Minimizes number of iterations
- Can be used by designers



# Tweaking performance

- Processing power varies greatly among mobile devices
- Not all effects are essential for game-play
- Blur, bloom, refraction & reflection are particularly expensive



# Test performance

- Determine device type
  - Thousands of devices
- Render graphics offscreen
  - May not reflect speed perfectly
  - Takes time



# Test performance

- Load assets before test
- Start with basics, add in more effects
- Measure frame rate
- Can be done in about one second



# Demo



# Mentioned tools

- SpriteBuilder
- CrazyBump
- Normal Mixer
- Sprite Lamp
- SpriteIlluminator



# More info and resources

[viktorious.com/gdc](http://viktorious.com/gdc)

- Slides from presentation
- Links to all tools
- Sample assets
- Links to more tutorials

[@viktorlidholt](https://twitter.com/viktorlidholt)