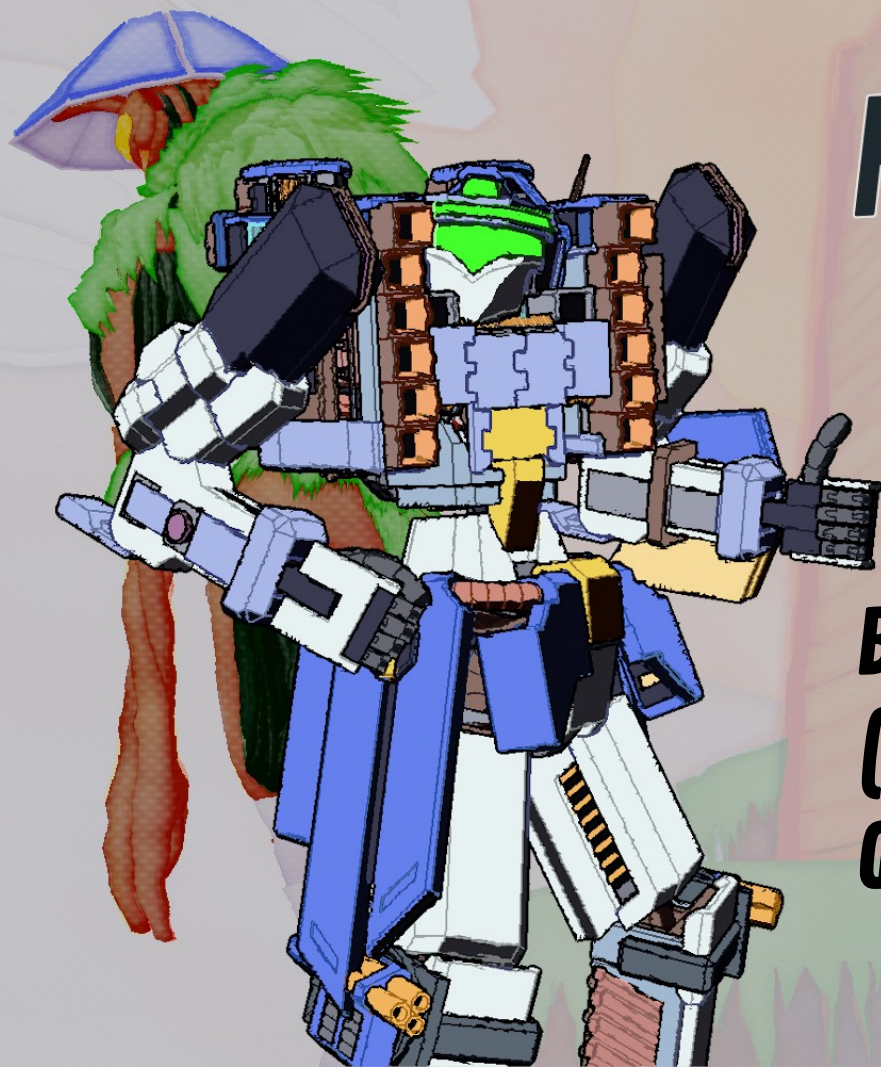




From *Watercolor* to *Mechs*

Stylized Rendering Pipelines in Godot

**By Panagiotis Tsiapkolis
(Panthavma)
Godot Con 2023**

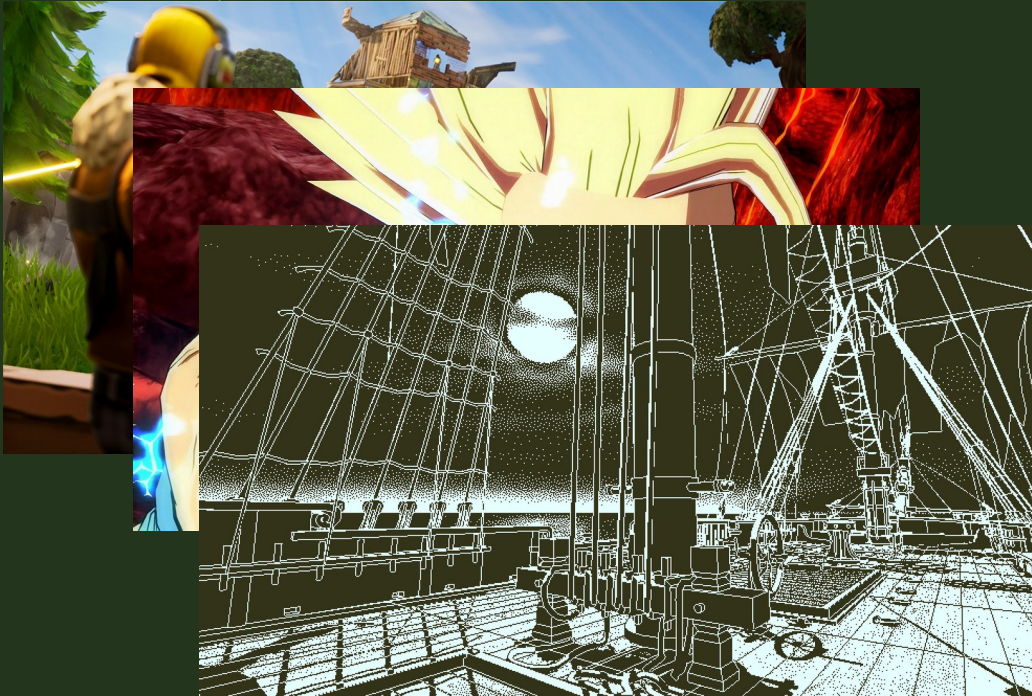
What is this talk about?

- ♦ Obstacles
- ♦ How to plan / solve
- ♦ How to put in practice
- ♦ NOT: How to make a specific style

What is Stylized Rendering?

NPR / Expressive Rendering

- ◆ Style



- ◆ Clarity



Photos vs Drawings



[T. Sanson]

Photo: Rules as laws

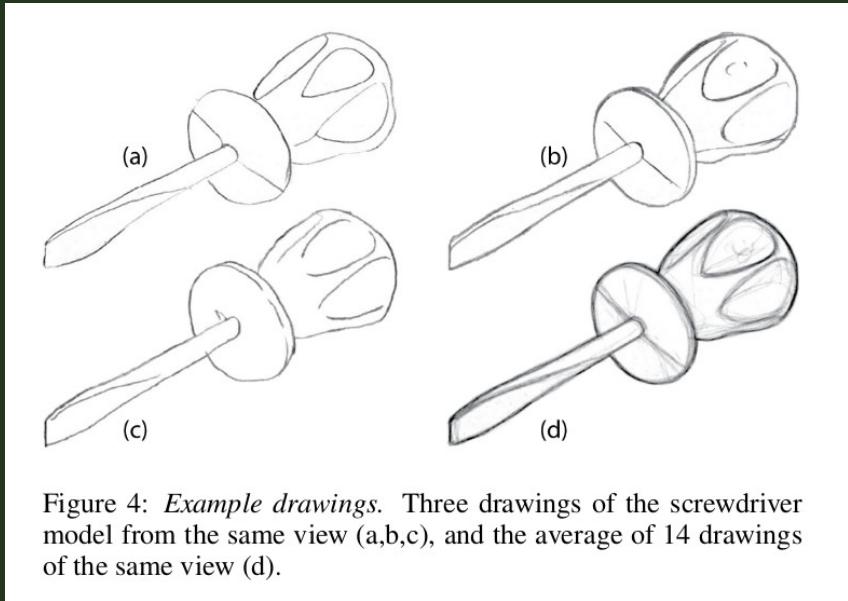


[Gundam Narrative]

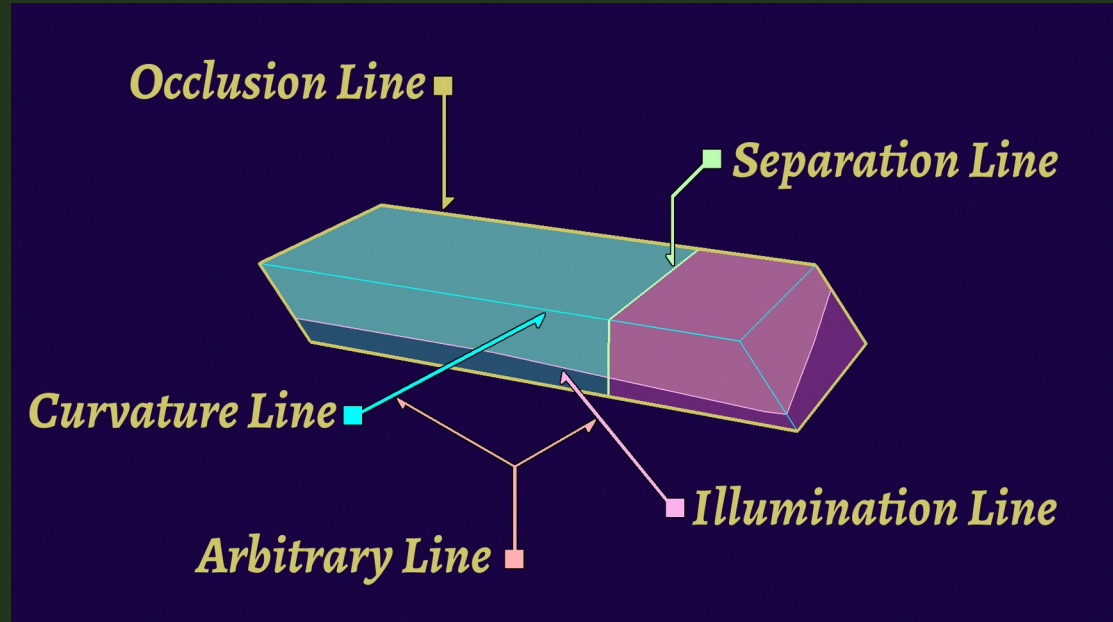
Drawing: Rules as guides

Problem of Control

- ♦ Rendering works like photos!



[Cole et al 2008]

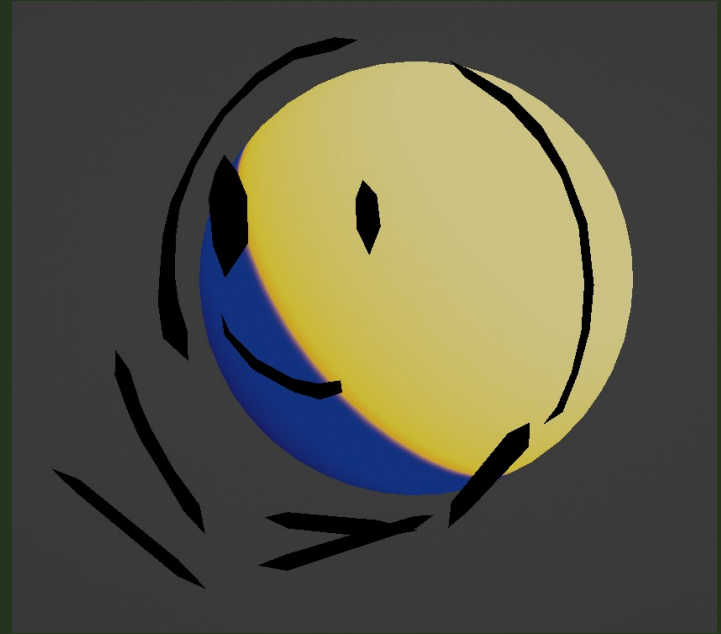


- ♦ Global info, semantics...
- ♦ Still an open problem!

Movement Problems



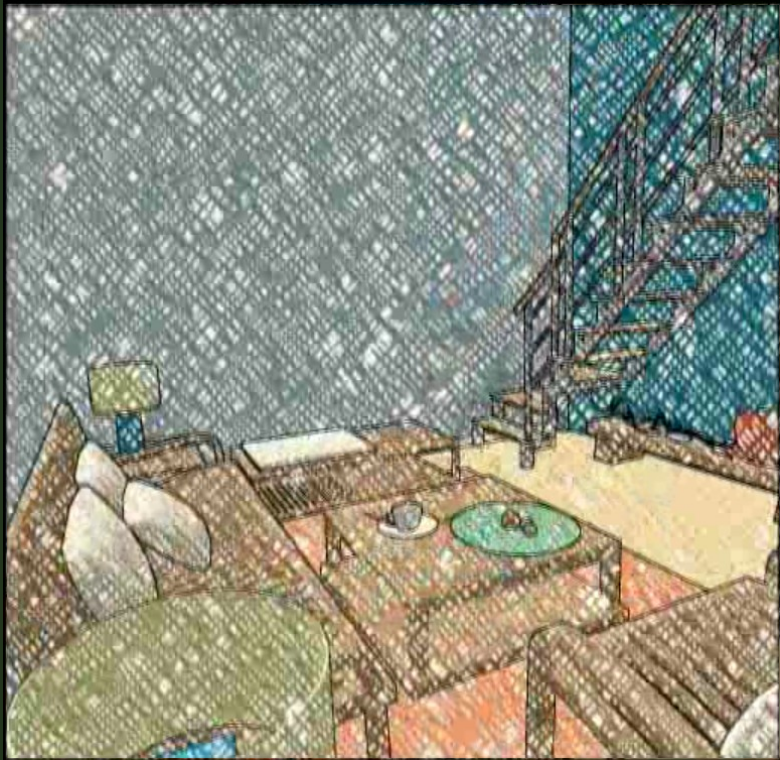
- ♦ Looks cool...



- ♦ But breaks!

- ♦ Some styles have it easier (anime)

Simple solutions are not enough!



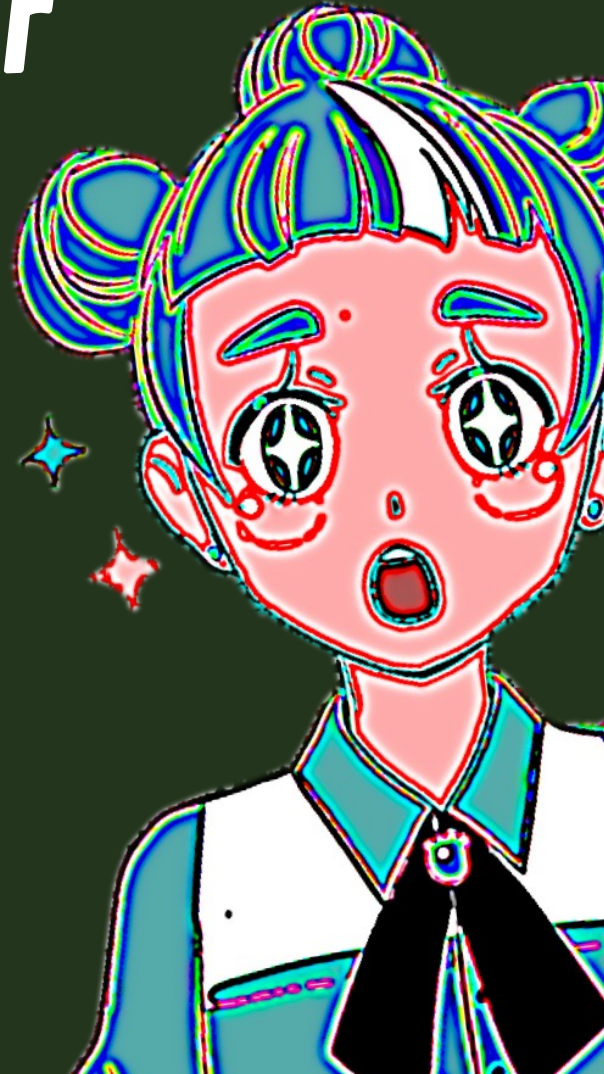
- ◆ Shower door effect



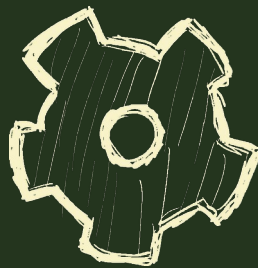
- ◆ Wallpaper Effect

5 Problems you'll encounter

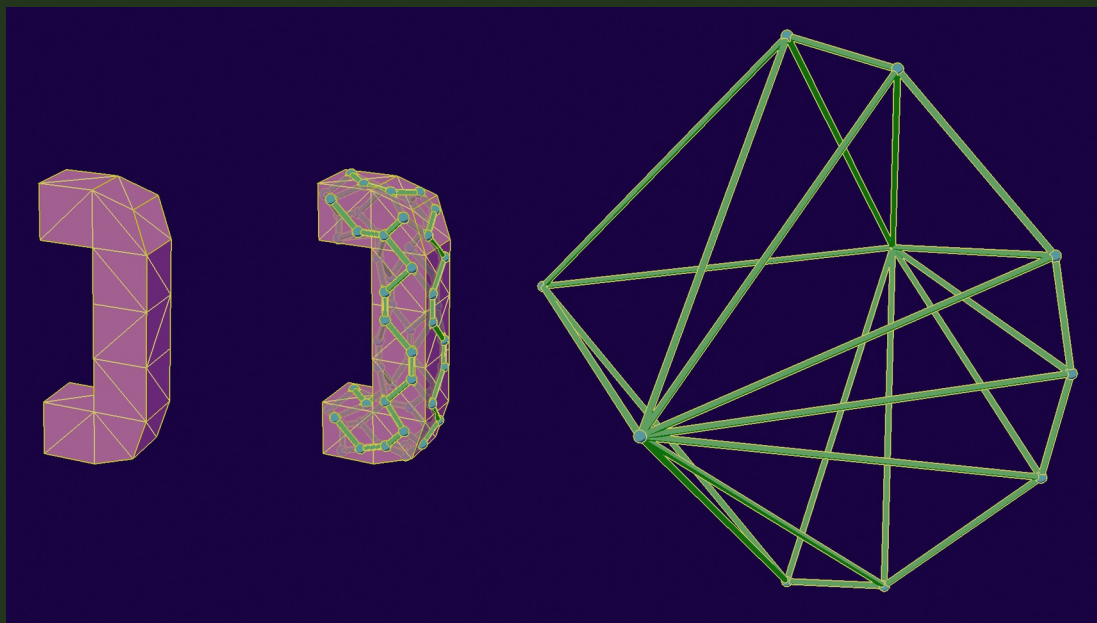
- ◆ General thought framework, not complete
- ◆ Good to get started



#1: Technology



- ♦ Just as technical as PBR!
- ♦ Hard to implement



***Solution:** Know your team and choose appropriately*

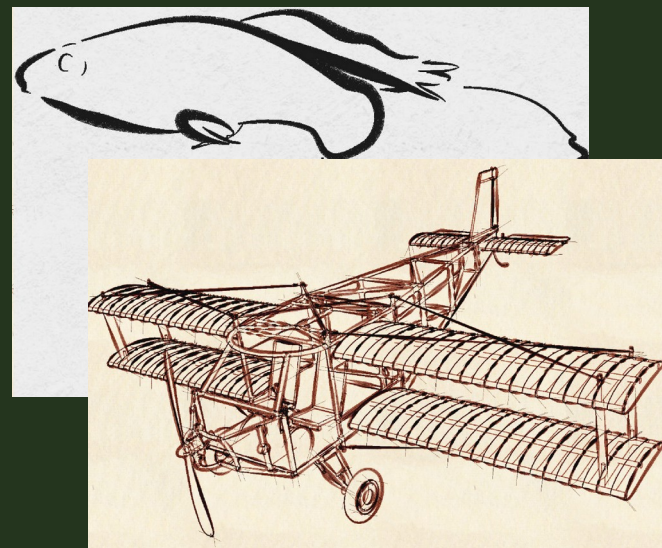
Line Rendering



- ♦ Inverted Hull
[Bomb Rush Cyberfunk]



- ♦ Screen Space
Extraction
[Kronian Titans]

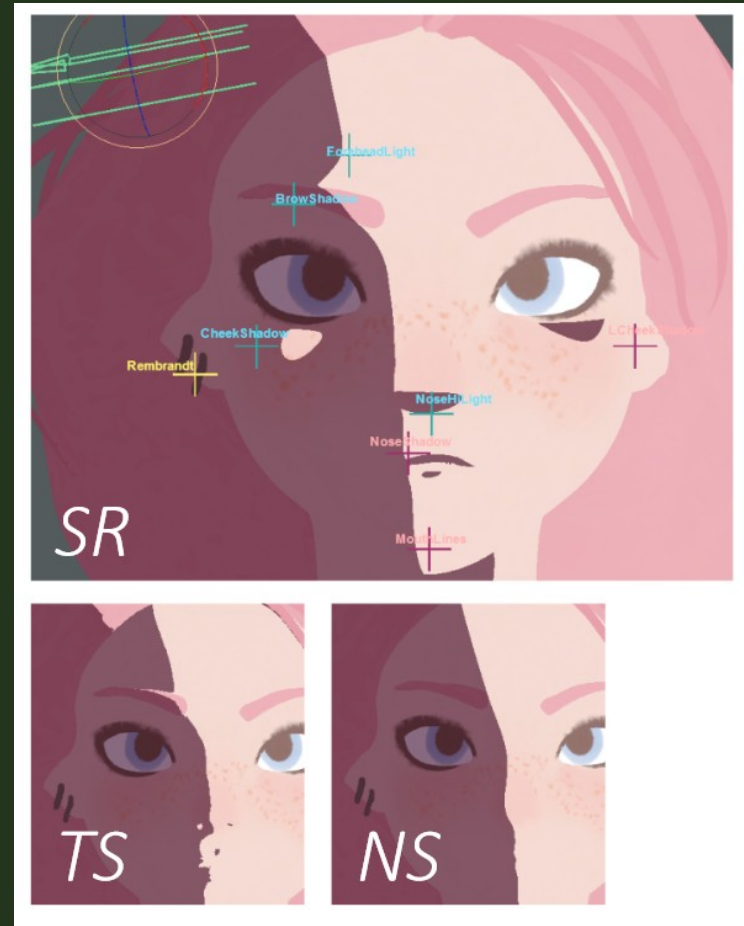


- ♦ Mesh contours
[Grabli et al 2010]

#2: Tools



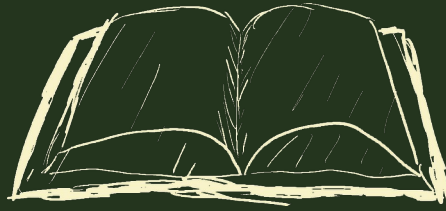
- ♦ Control is hard!
- ♦ Pipelines focused on PBR
- ♦ Specific controls still an open problem
- ♦ Often needs custom work



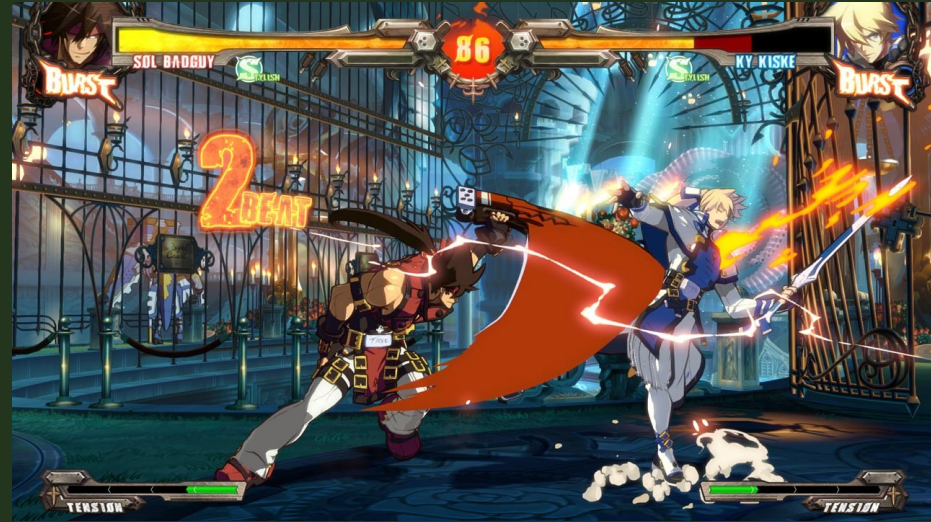
[Petikam et al 2021]

We'll come back to that later

#3: Knowledge



- ♦ Not super widespread!
- ♦ Very different perspectives
- ♦ Rare references overshadow others
- ♦ Research is hard to read and implement



[Guilty Gear Xrd]

***Solution:** Read from different sources (blogs, artists and programmers)*

#4: *Organization*

- ♦ The right people at the right time!
- ♦ Preproduction is critical:
 - Fast Iteration needed
 - Creating the tools



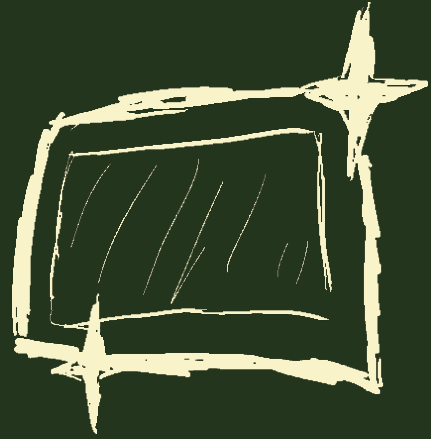
***Solution:** Schedule it, with your teams or external collabs*

#5: *Production*

- ♦ Additional risk on two fronts!

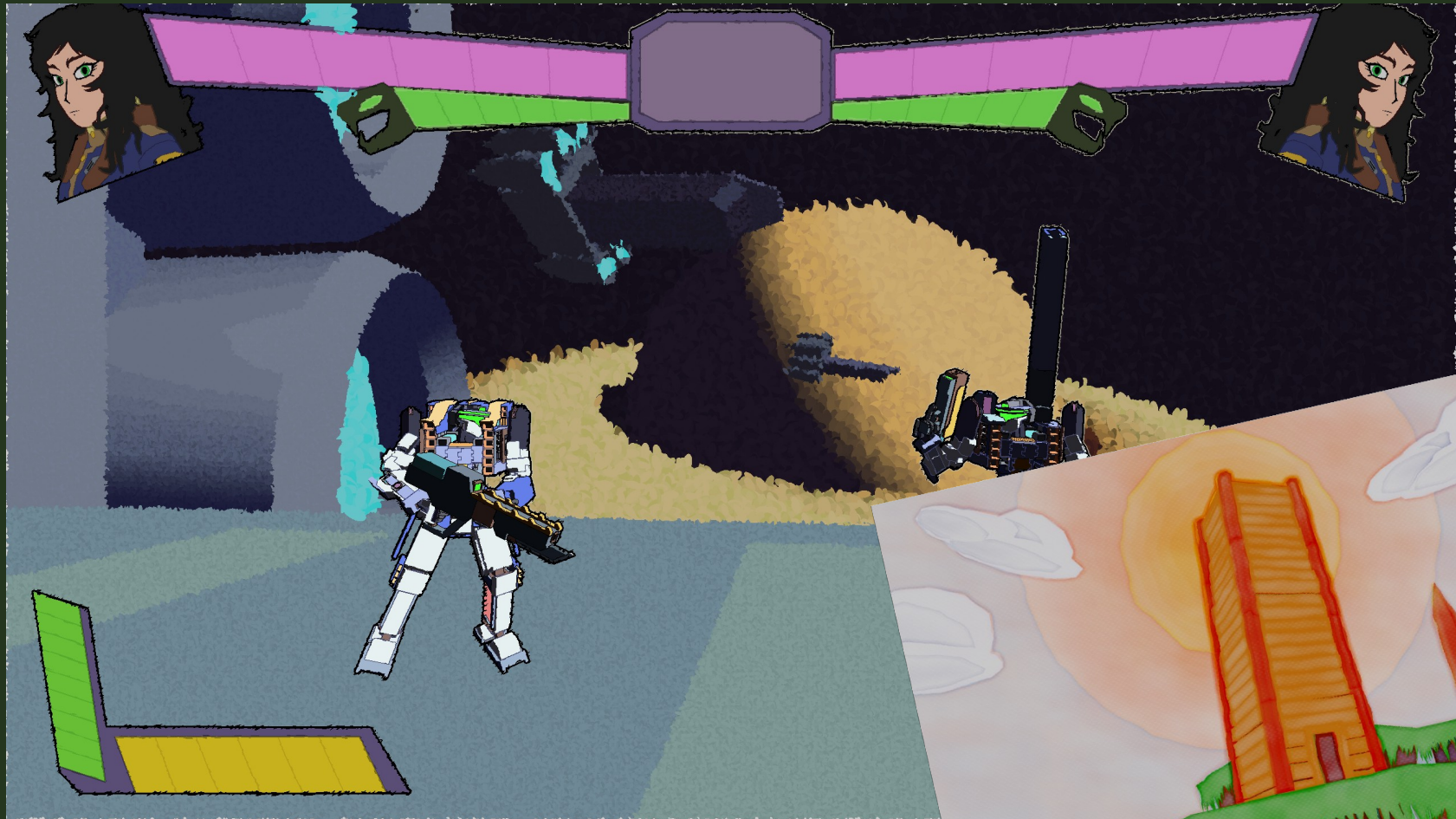


- ♦ **TIME:**
No experience or standards
 - BUT! Can be an acceleration factor



- ♦ **MARKETTING:**
Style is divisive
 - BUT! Can be a unique selling point

Let's see it in practice! [Kronian Titans]



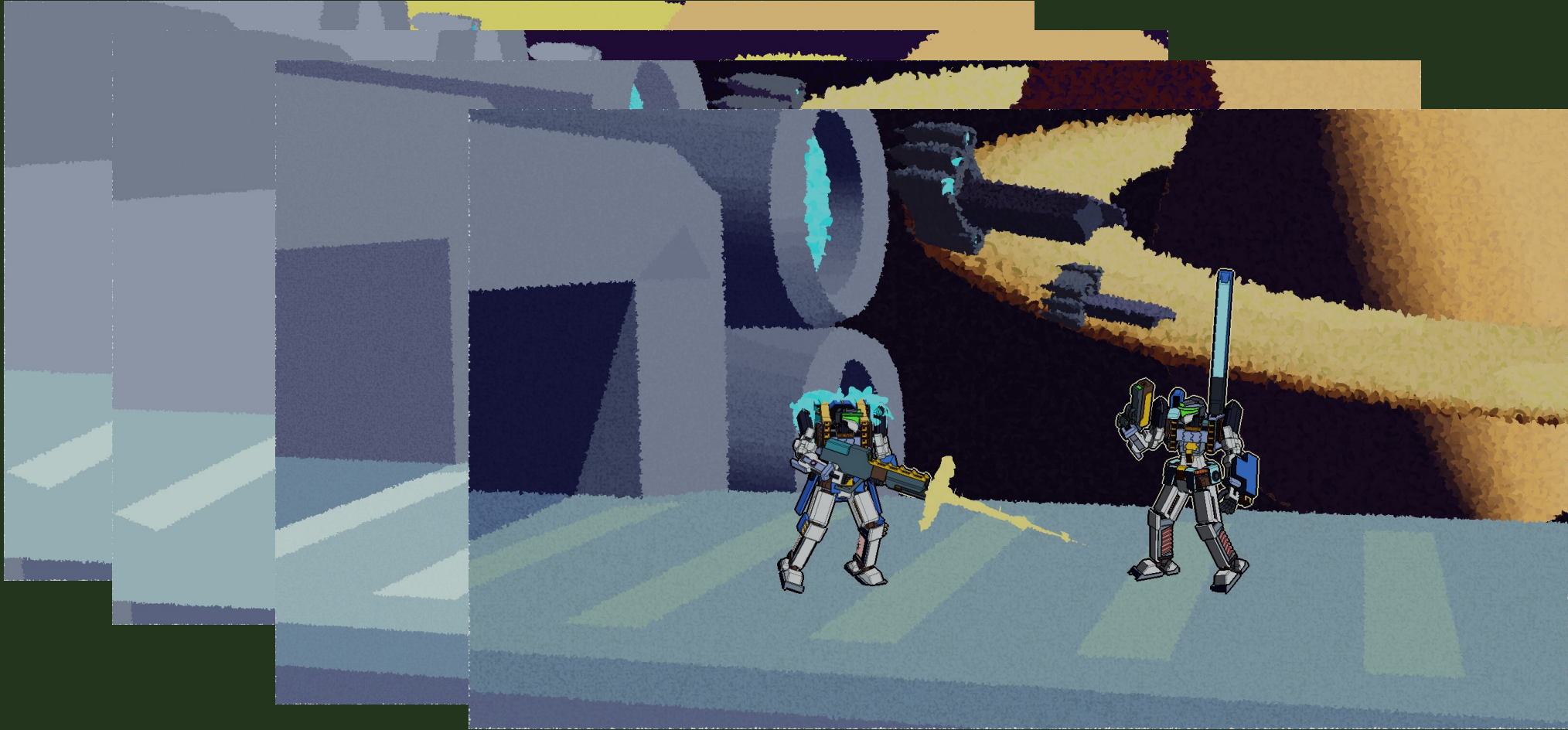
Define your objectives!

- ♦ **WHO IS IN THE TEAM**
 - Me, technical but not that artistic
- ♦ **GAME PRIORITIES**
 - Must separate front and back, readable with speed
- ♦ **PROD PRIORITIES**
 - Art bottleneck: need to leverage rendering

Iterations...



Iterations...

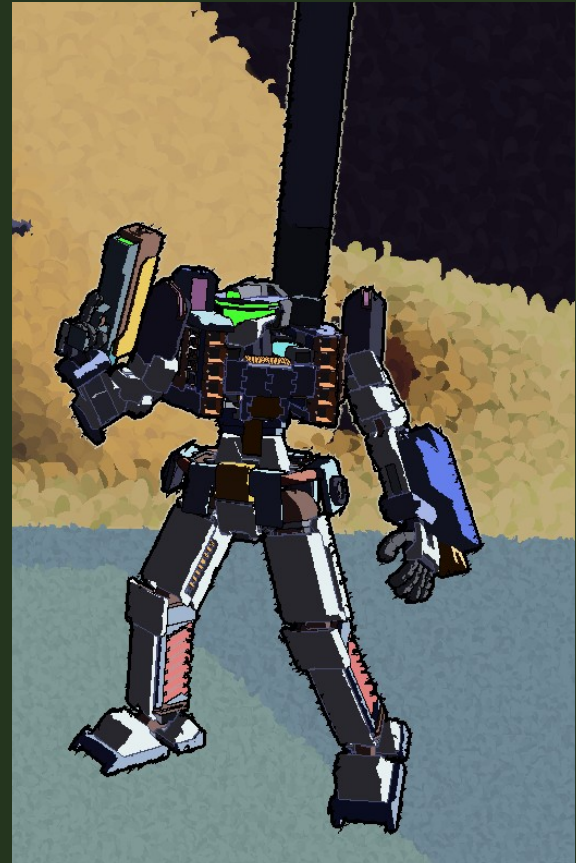
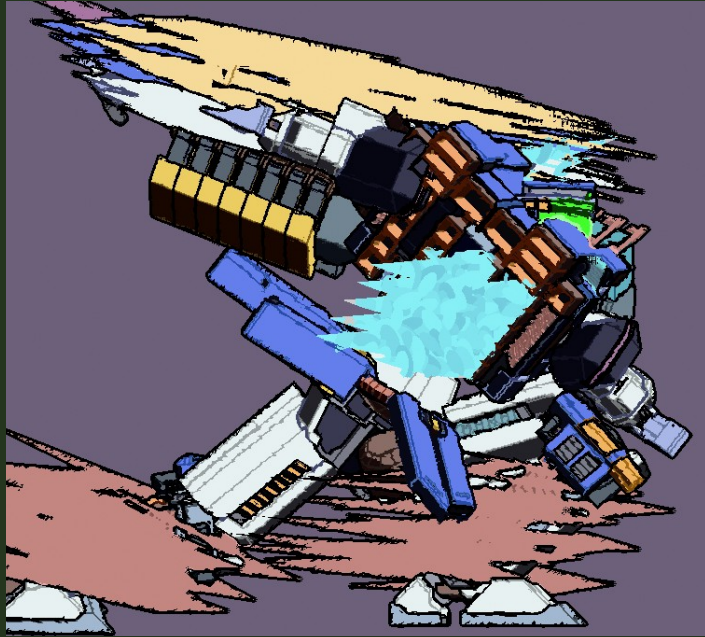


Iterations...

- ♦ Tricky, need to program



Resulting Direction



- ♦ Heavy hierarchical lines
 - My drawing style!

- ♦ Smears in animation

- ♦ Different style and more abstract with distance

How much to change rendering?

- ♦ A little
 - Easier and integrated
 - More limited
- ♦ Hijacking
 - Twist the features for your use
 - Bleeding edge
- ♦ Custom Renderer
 - Most features (tech & tools)
 - Long and hard to make

Use a little bit of rendering

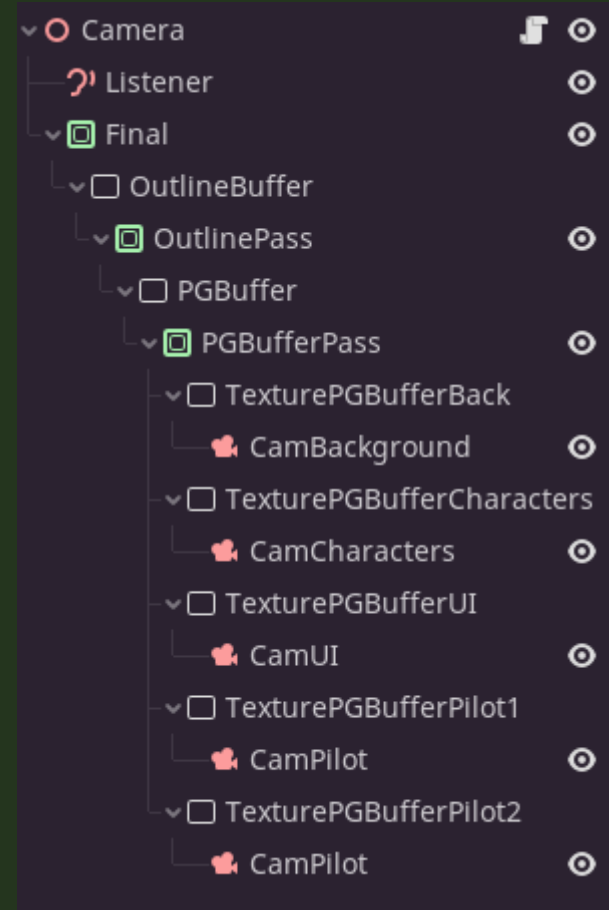


- ♦ Apply it at specific places
 - Shading, global adjustment...
- ♦ Precompute / Prerender
- ♦ Can be a strong accelerator!



Hijacking Godot 3: Limits

- ♦ All light data in first pass (forward renderer)
- ♦ One output only (but can chain passes)
- ♦ Passes have a cost due to viewports / render

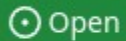


Sidenote: Godot 4?



- ♦ More features for PBR
- ♦ General philosophy doesn't synergize
- ♦ Some discussion of relevant improvements

Implement a Rendering Compositor #7916



reduz opened this issue on Sep 29 · 28 comments



reduz commented on Sep 29 · edited ▾

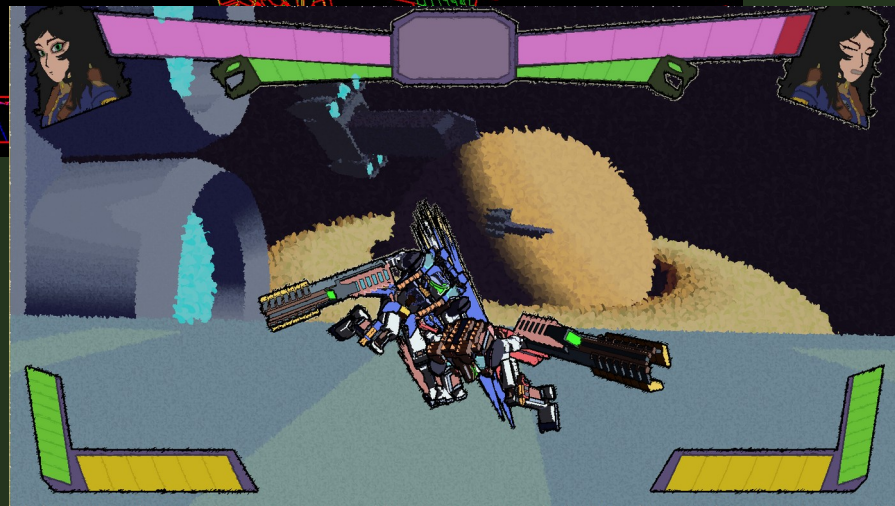
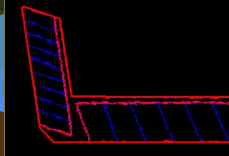
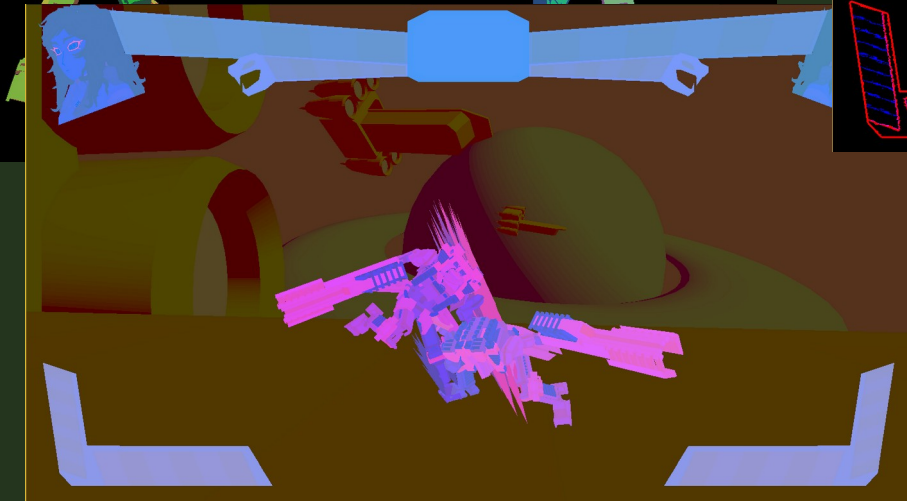
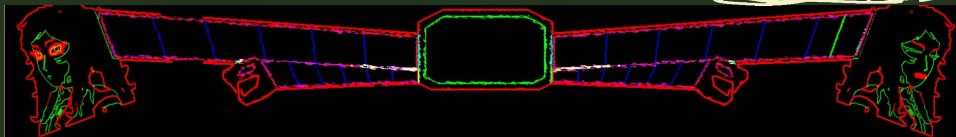
Hijacking Godot 3: How to



- ♦ Choose algorithms that work in those limits
- ♦ Choose the data to pass
- ♦ Can link some values through GDScript
- ♦ /!\ No access to some effects!
 - (Transparency, post process)

My Pipeline (Pseudo-Deferred)

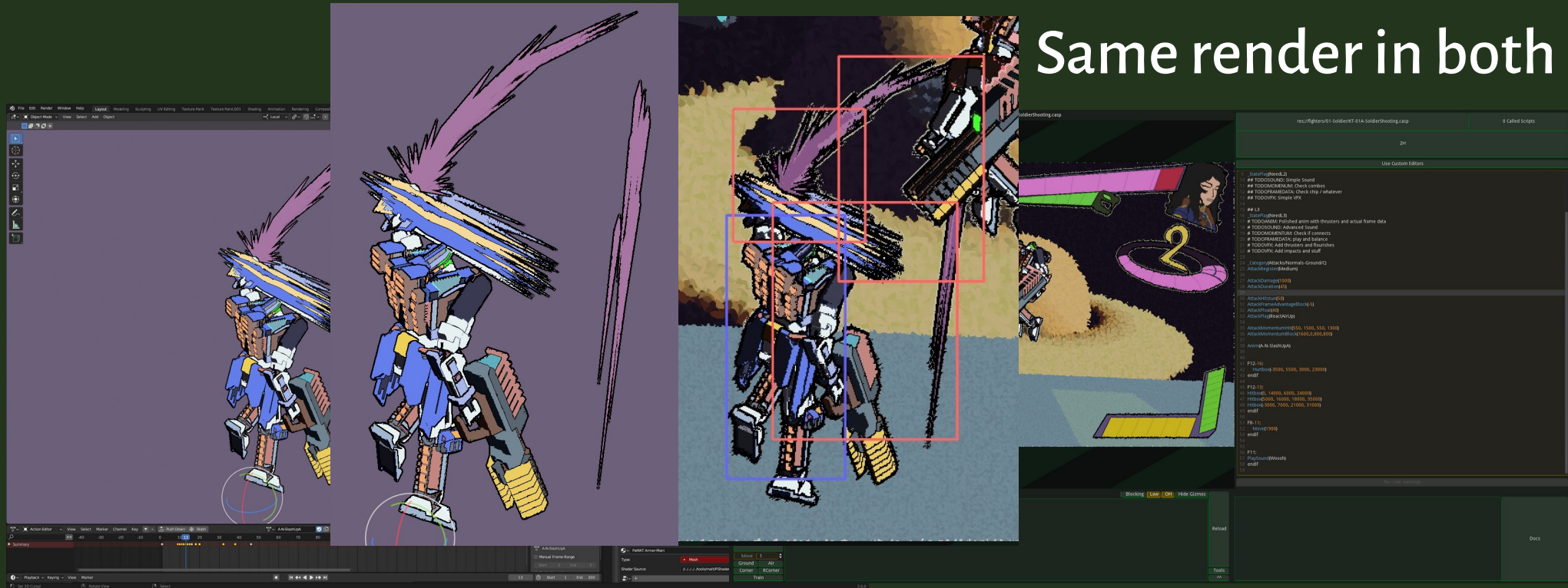
Extra



Tool Chain with Blender / Malt



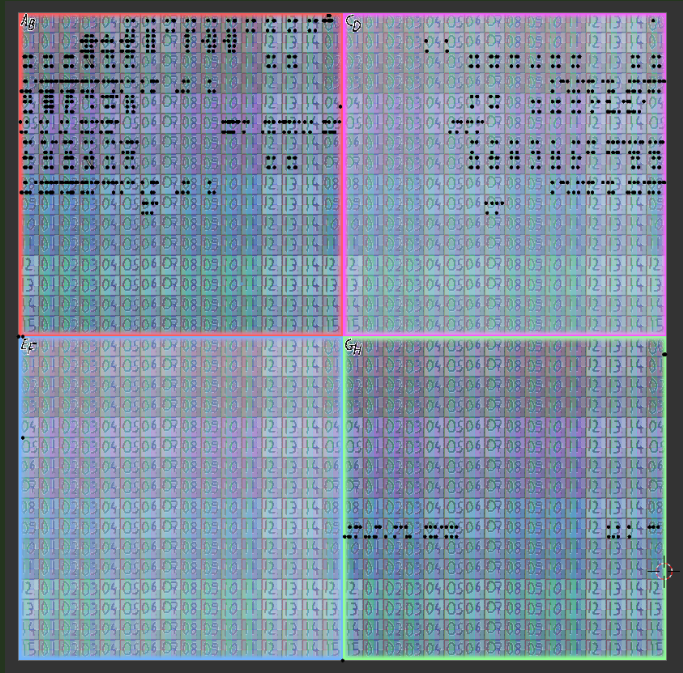
Same render in both !



- ◆ Blender + Malt
- ◆ Export shading data in assets = direct use

Passing material data

Link

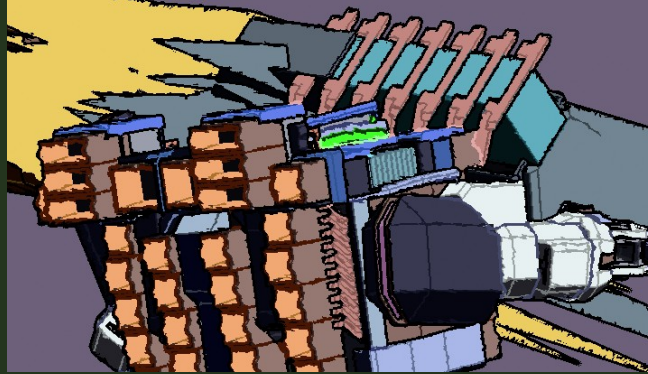
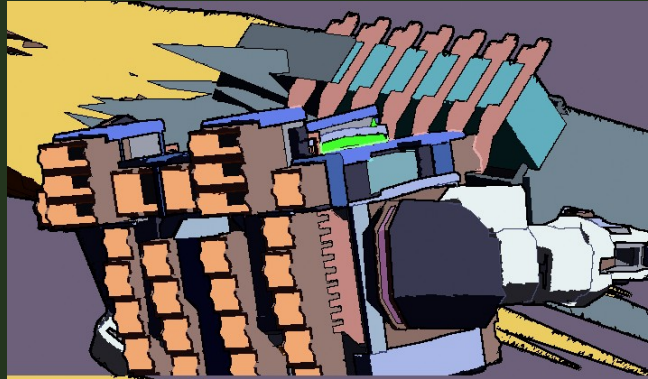


- ♦ Data in vertices (UV)
 - My method: Material ID
- ♦ 16x256 mats in 64x64 texs
 - Created automatically

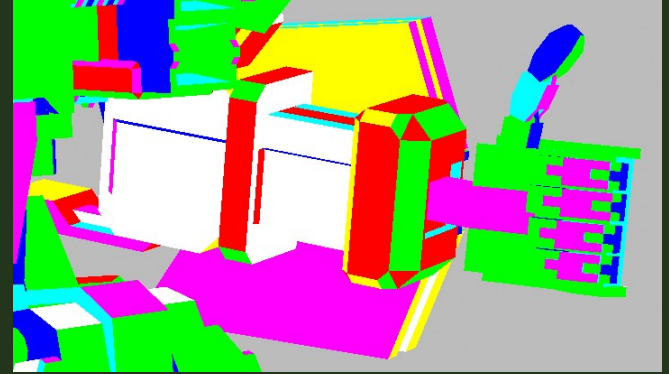
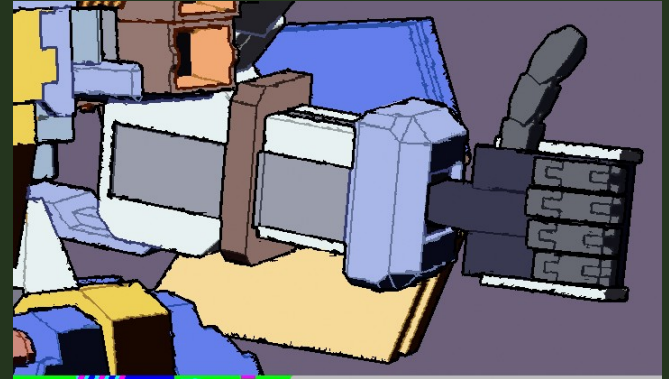
Choose your controls: Kronian



- ♦ Color under lighting conditions

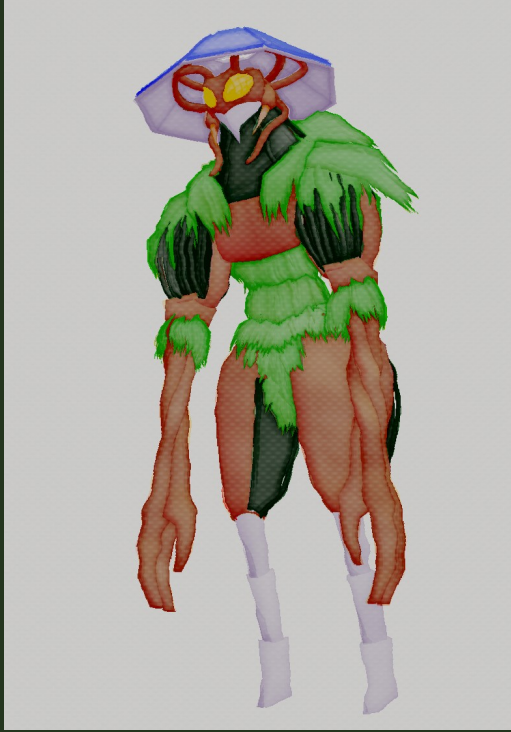


- ♦ Line color and style (far vs near)



- ♦ Line separation (ID based lines)

Choose your controls: Watercolor



- ♦ Pigment density & Edge darkening



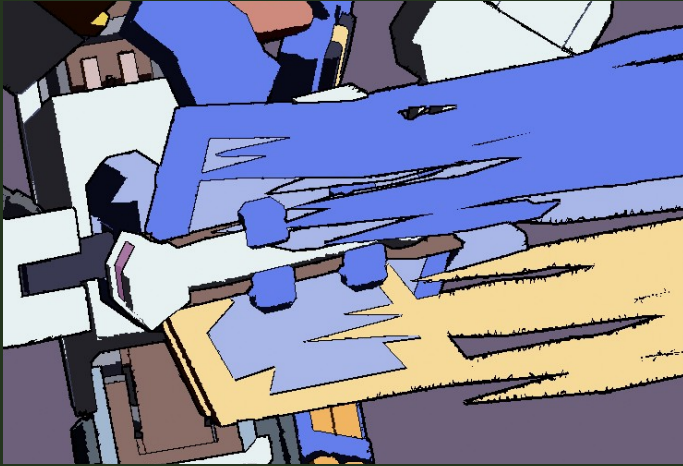
- ♦ Deformations per group



- ♦ Color and zones per shading

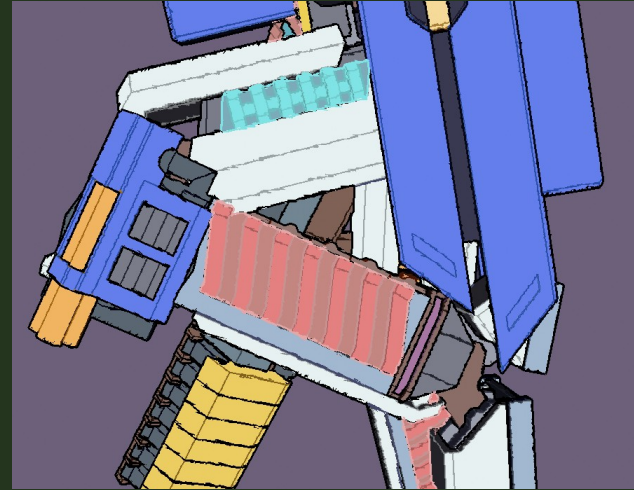
Model Construction

- ♦ Controls should take all levels into account



- ♦ Smears use SSE to merge

- ♦ Animation emphasises those elements



- ♦ Rig has automation

Adapt to your team!

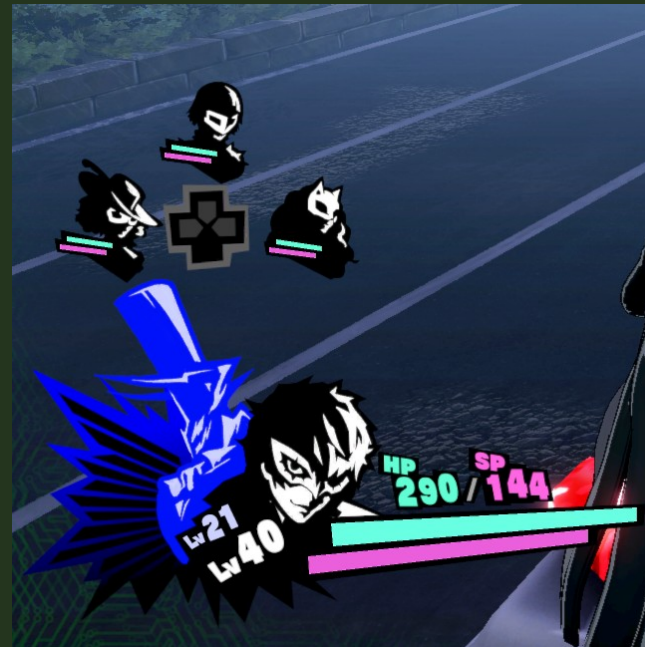
- ♦ More artistic?
 - Less control automation
 - Verifications to avoid errors
- ♦ More technical?
 - More automation of controls
 - Shader driven = more leverage

Use it elsewhere!



[Persona 5]

- ◆ Menus
 - Can often benefit from style and clearer info



[Persona 5 Strikers]

- ◆ UI
 - Icon automation to go faster

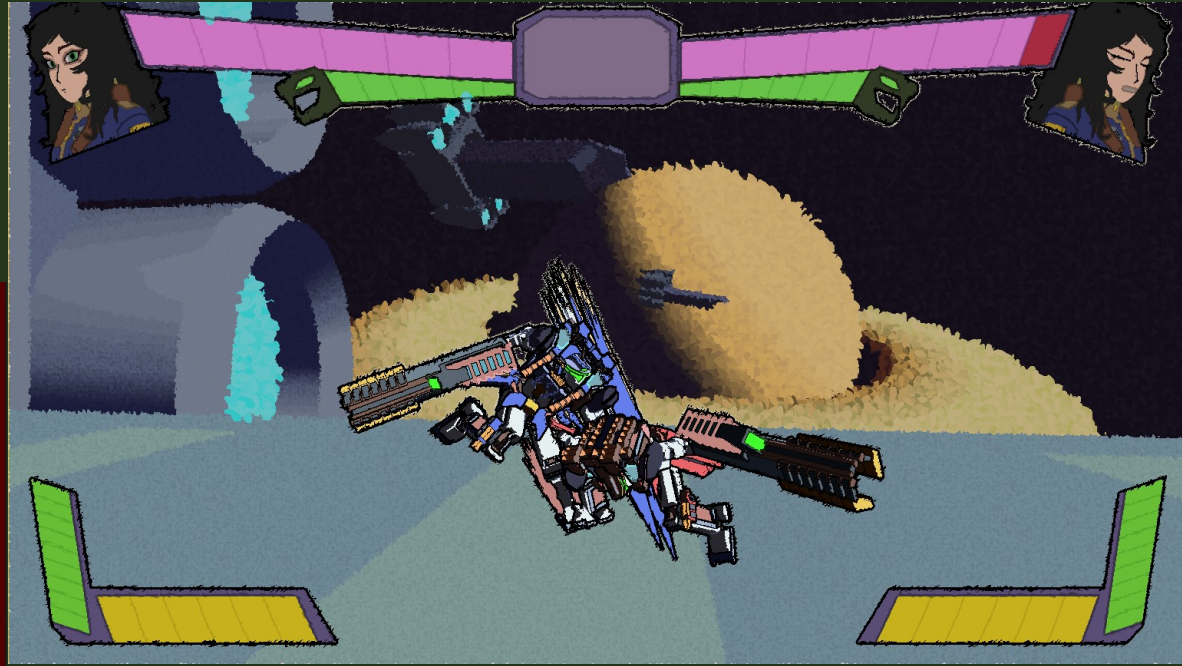
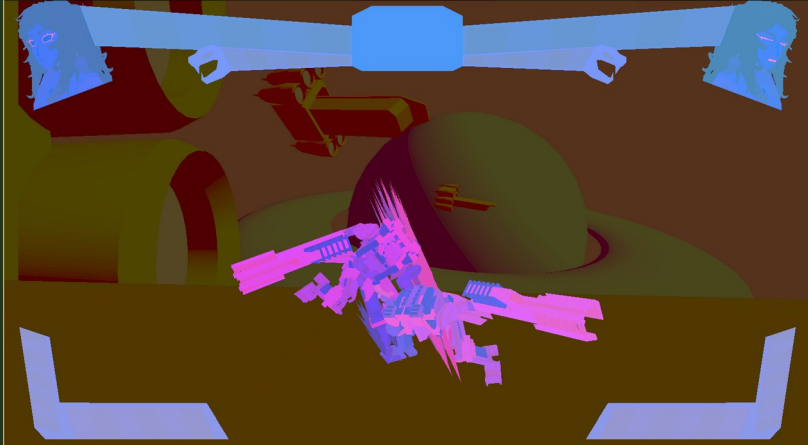
Thanks for listening!

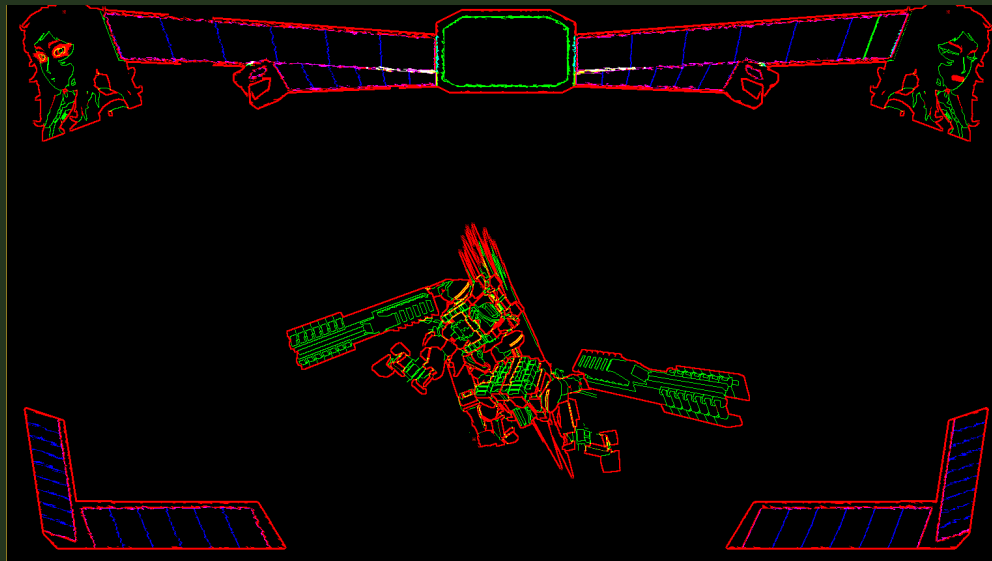
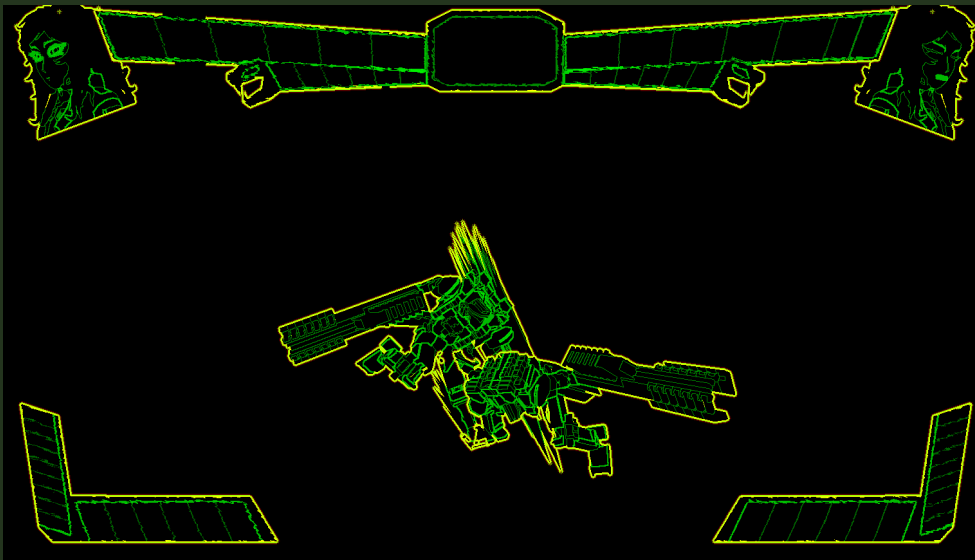
- ♦ Plan it from the start
- ♦ Needs specific skillsets
- ♦ Often custom tools
- ♦ Can be an opportunity!
- ♦ Companion page ->
- ♦ Extra slides
 - Line extraction, Kronian, Watercolor, GBC Rendering



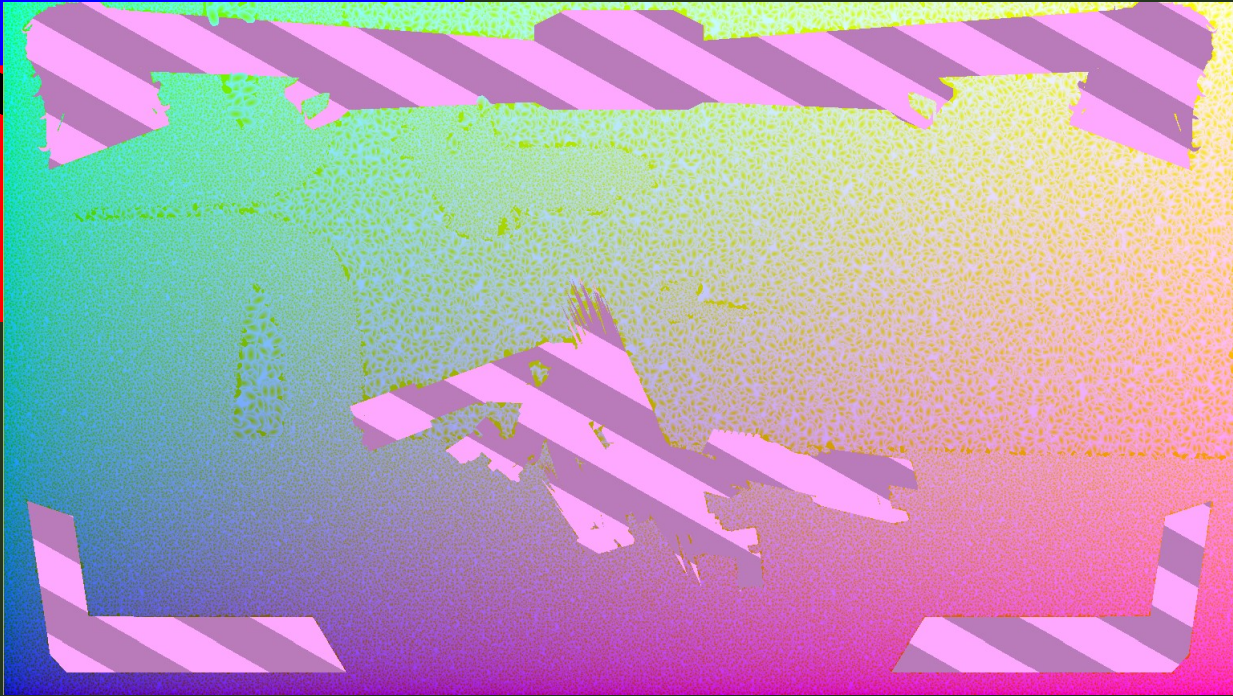
<https://panthavma.com/articles/godot-con-2023>

Kronian Titans Buffers





- Camera
- Listener
- Final
 - OutlineBuffer
 - OutlinePass
 - PGBuffer
 - PGBufferPass
 - TexturePGBufferBack
 - CamBackground
 - TexturePGBufferCharacters
 - CamCharacters
 - TexturePGBufferUI
 - CamUI
 - TexturePGBufferPilot1
 - CamPilot
 - TexturePGBufferPilot2
 - CamPilot

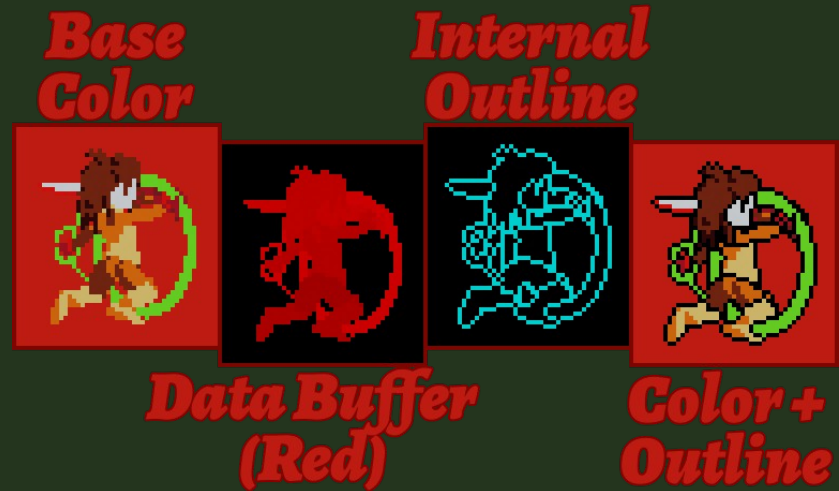


Watercolor





Prerender



Dither Color Buffer



Data Buffer



Alpha Mask



Dithering



***Outline
Detection***



Base Color



***Outline
Internal***



***Outline
External***



***Shaded
Color***

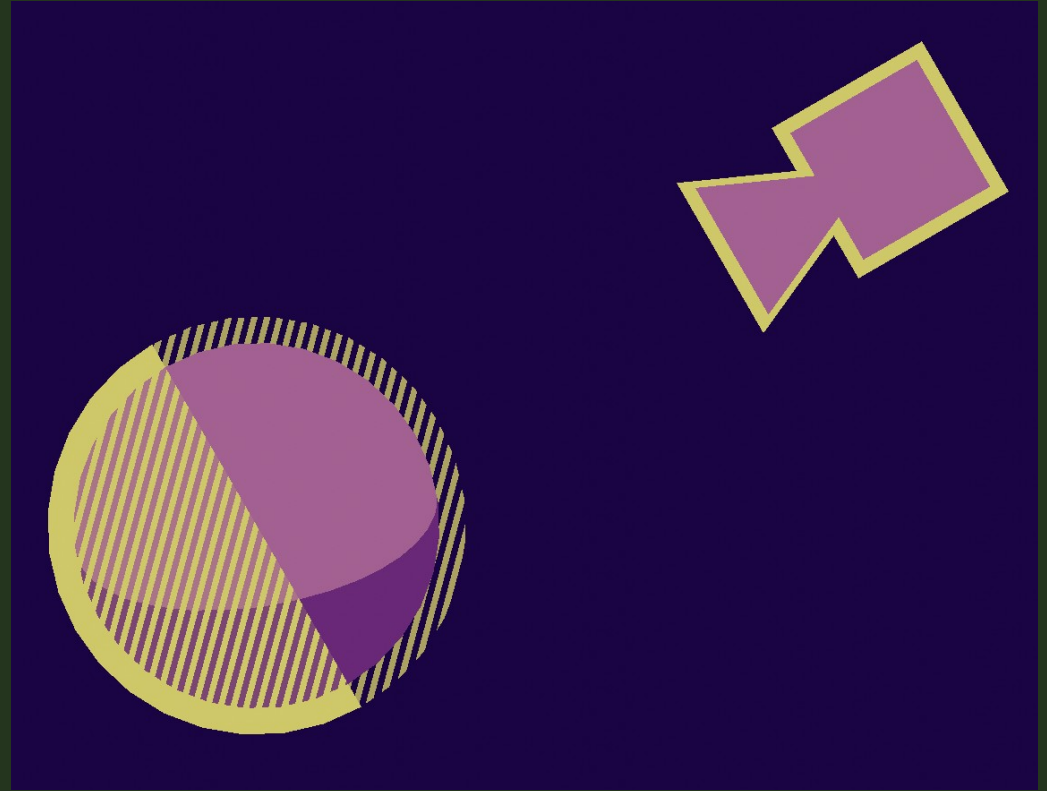


***Final
Outline***

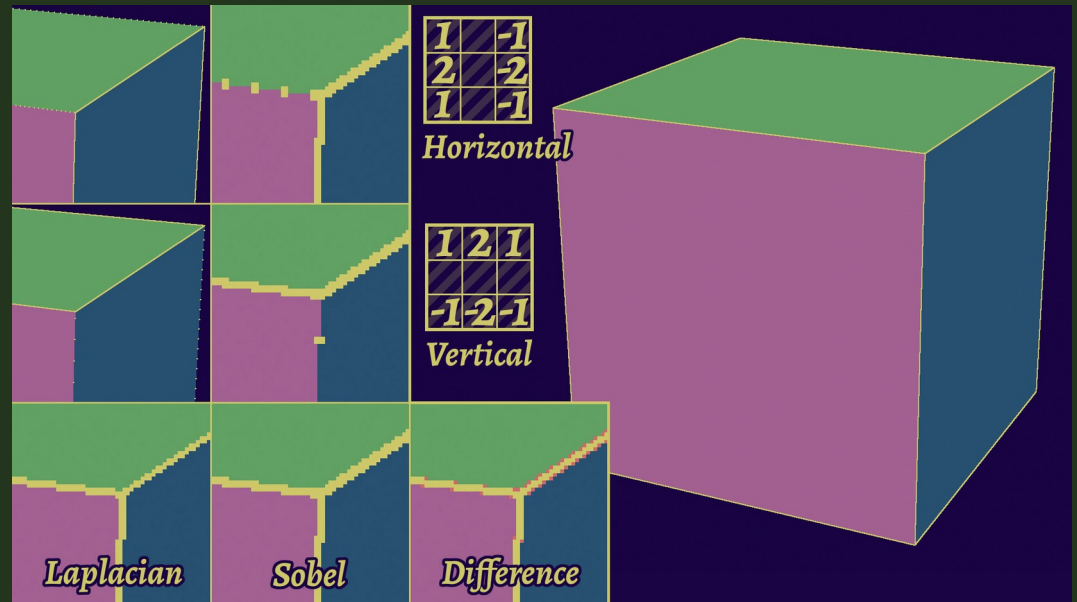
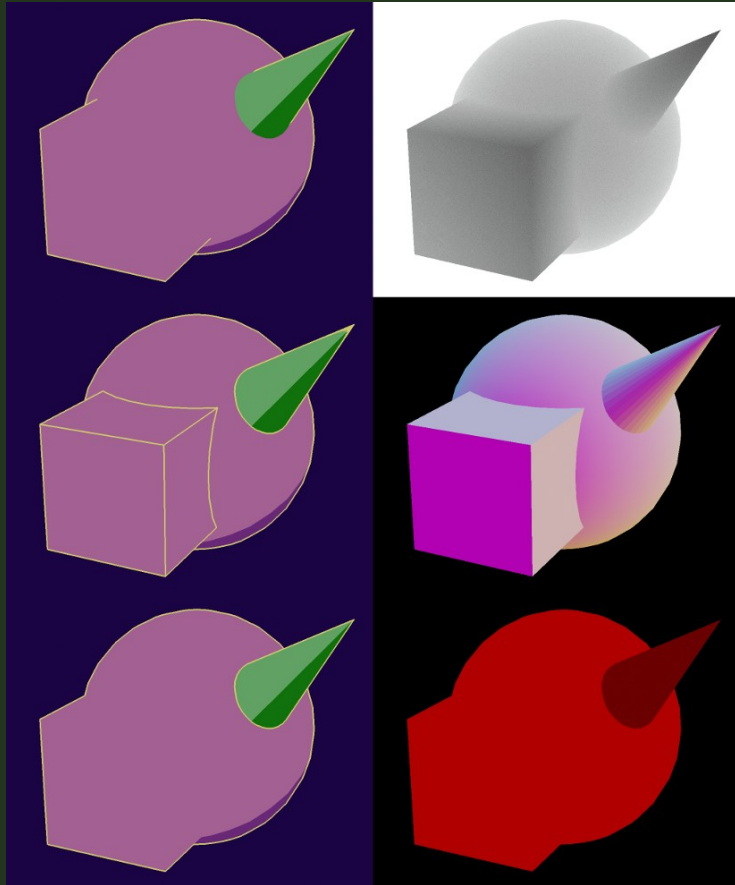


Final Render

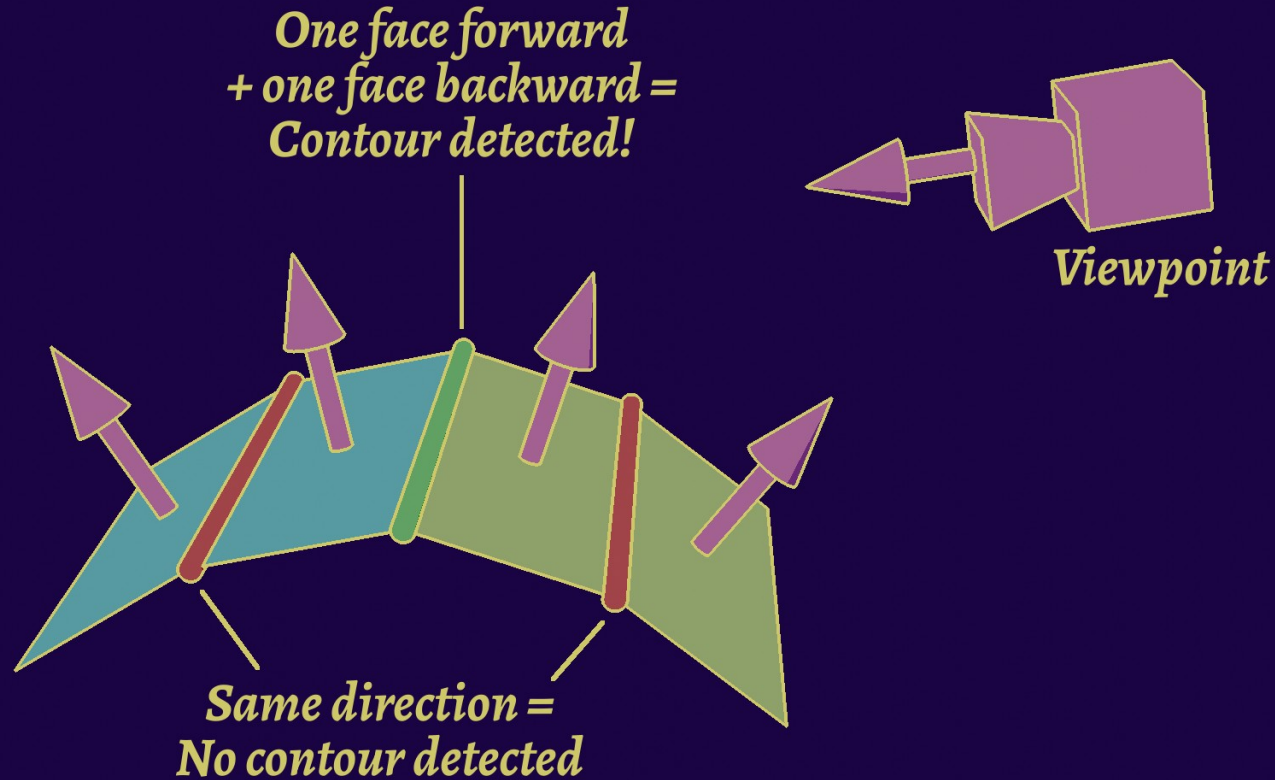
Inverted Hull



SSE Lines



Mesh Contours



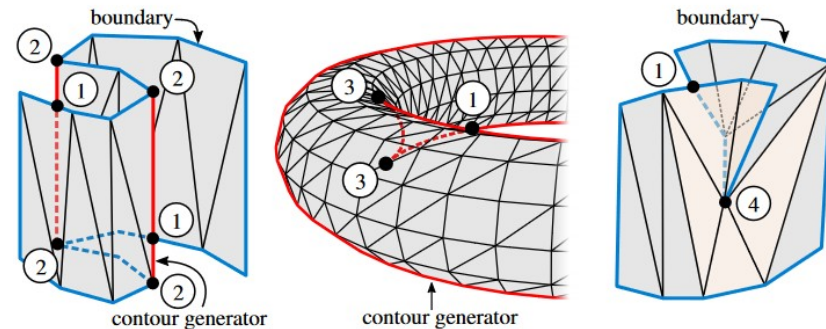
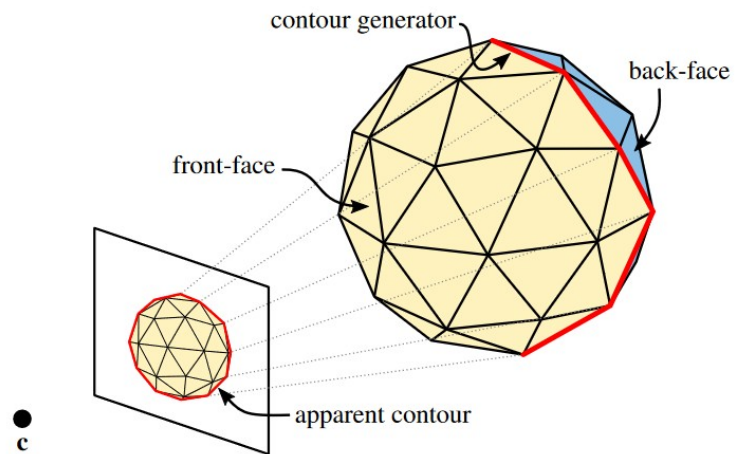


Figure 4.2: Singular points — From the camera viewpoint, (1) T-junctions at image-space intersections, (2) Y-junctions between a contour generator and two boundary edges, (3) contour generator curtain folds, (4) boundary curtain-folds. Contour generator edges are drawn in red, and boundaries in blue. Curtain folds are visualized in more detail in Figure 4.3.

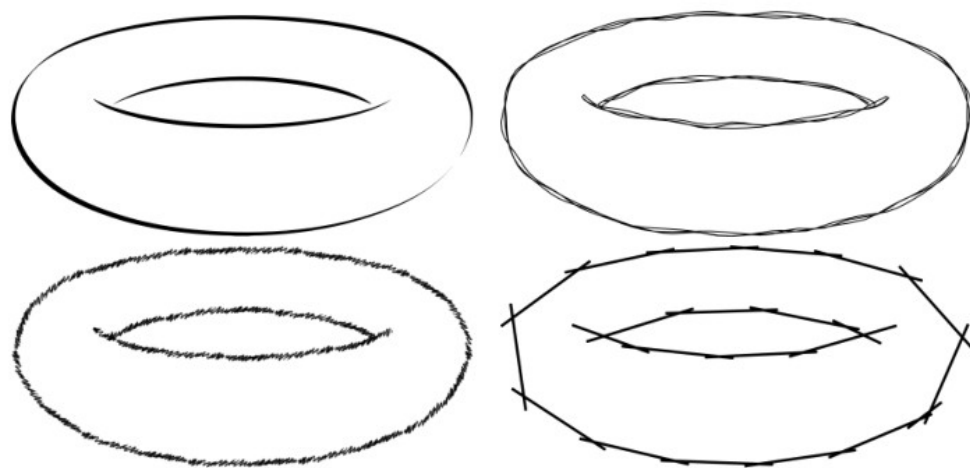


Figure 9.4: Stroke style — Various stroke styles applied to the contours of Figure 7.6b using Blender Freestyle. From left to right, top to bottom: calligraphy, sketchy overdraw, textured scribbles and guiding lines.

