Next-Gen Characters: From Facial Scans to Facial Animation

John Hable

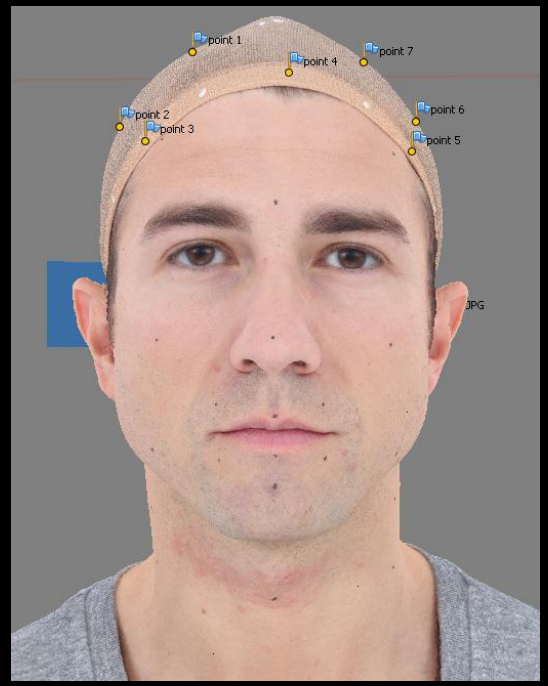
FilmicWorld.com

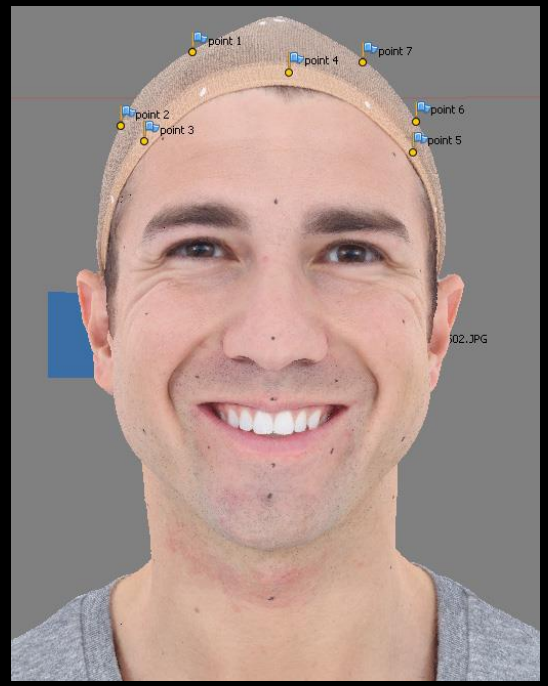
@FilmicWorlds

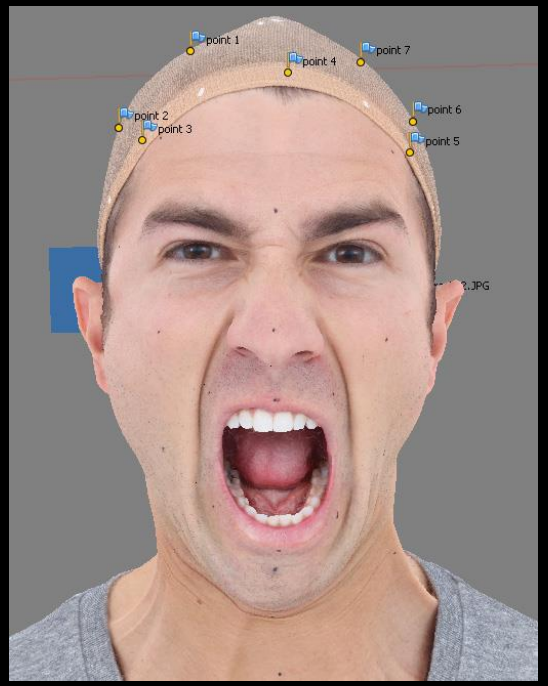
Movie:

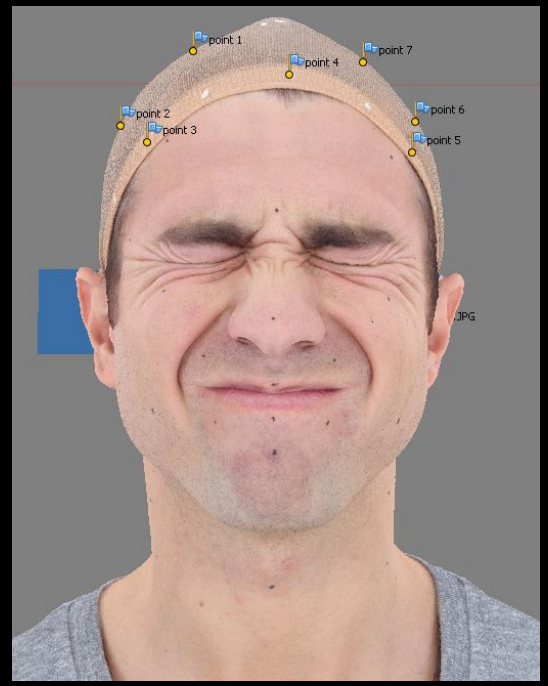
Facial Scans

- Raw scans look great
- How do we rig and animate it



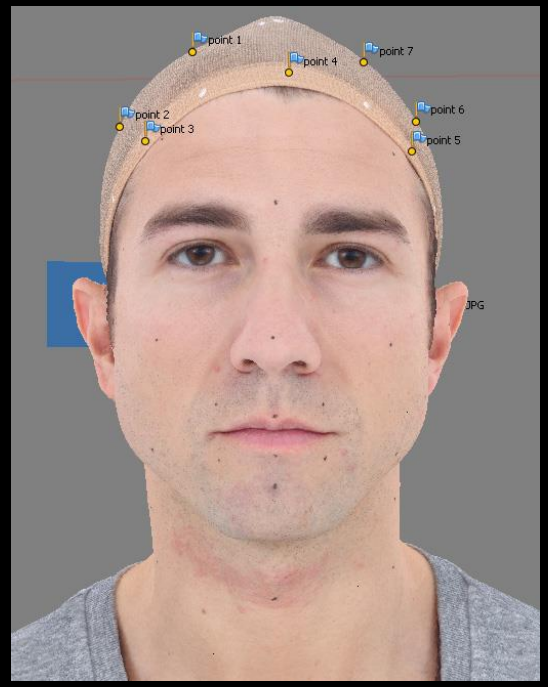


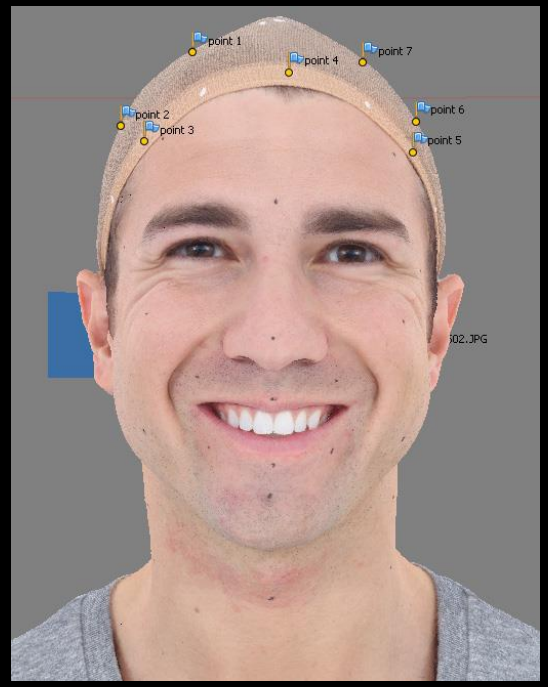


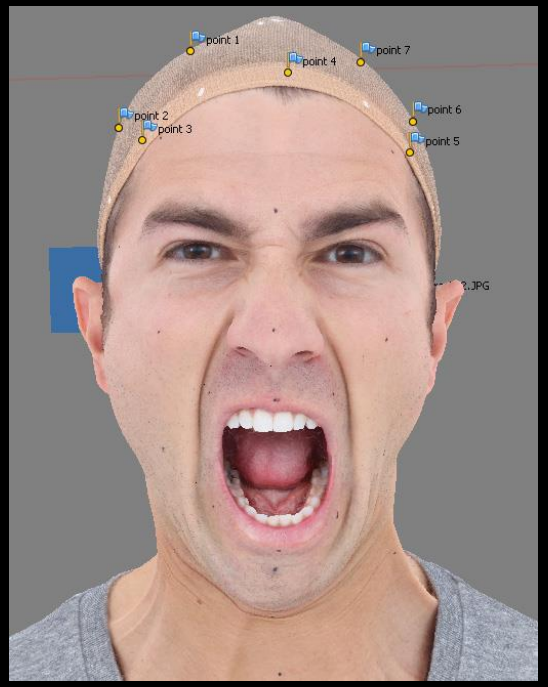


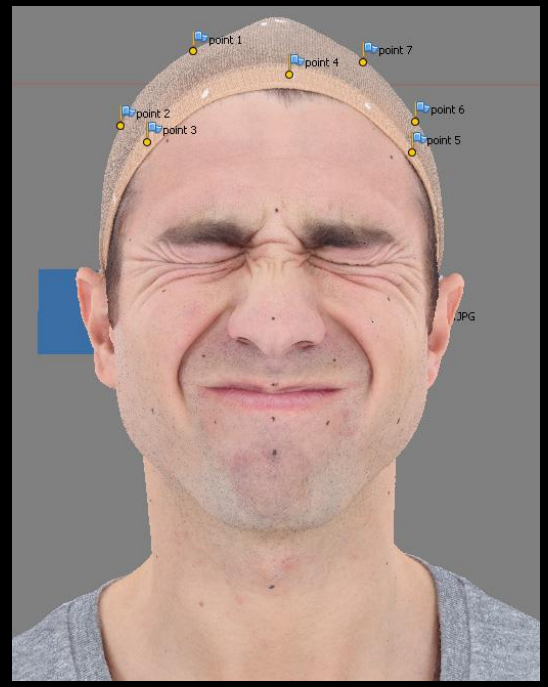
Rig

- Problem: Define a rig that matches these deformations
- Just geometry?
  - No: Need diffuse animation too

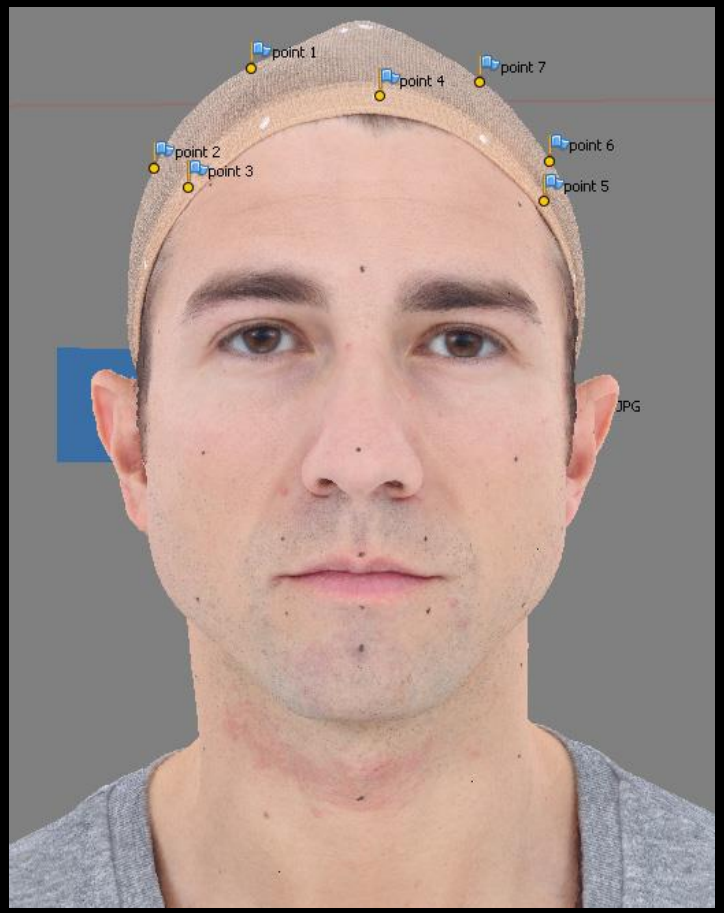


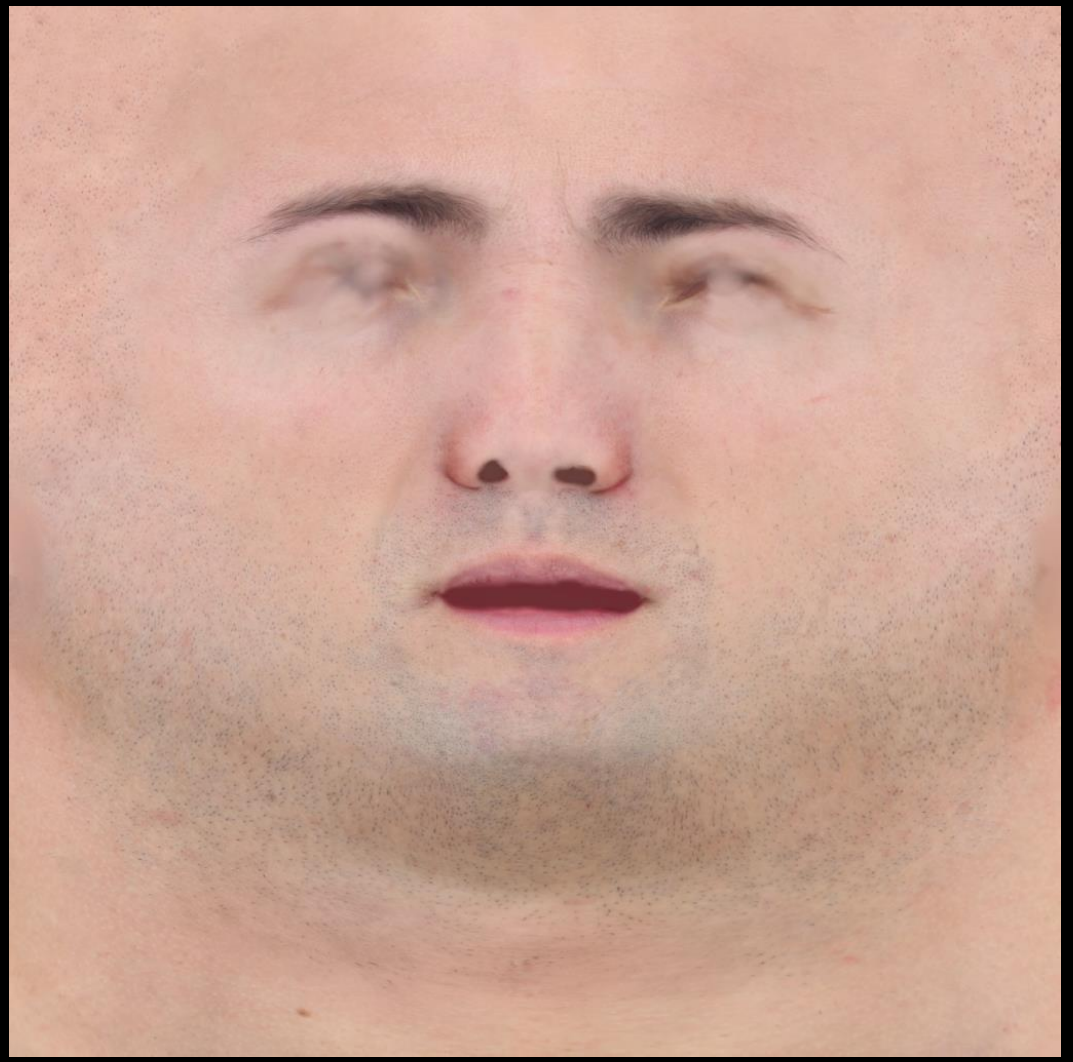




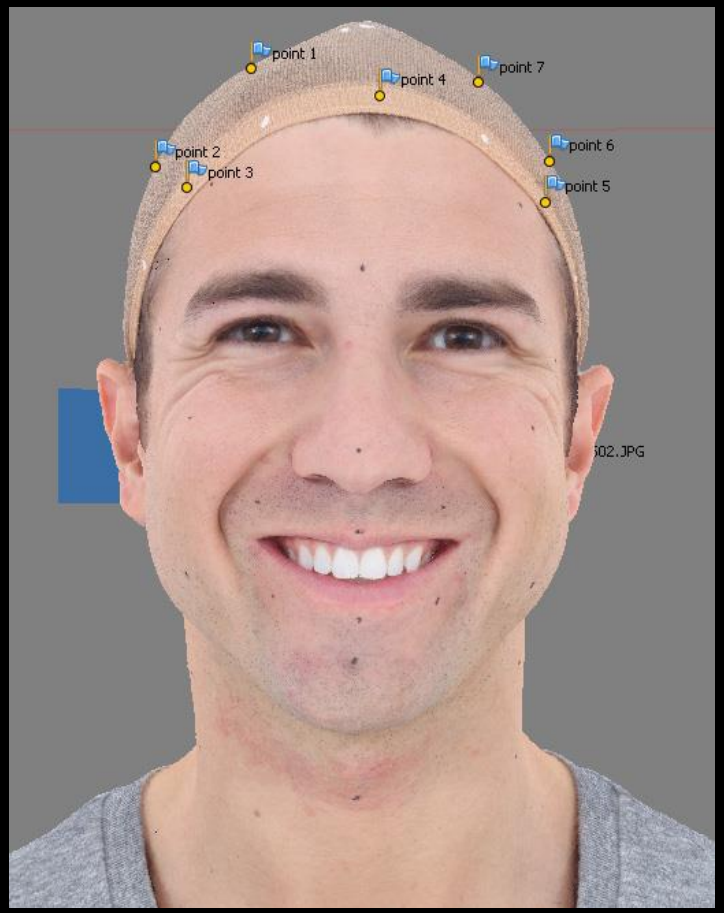


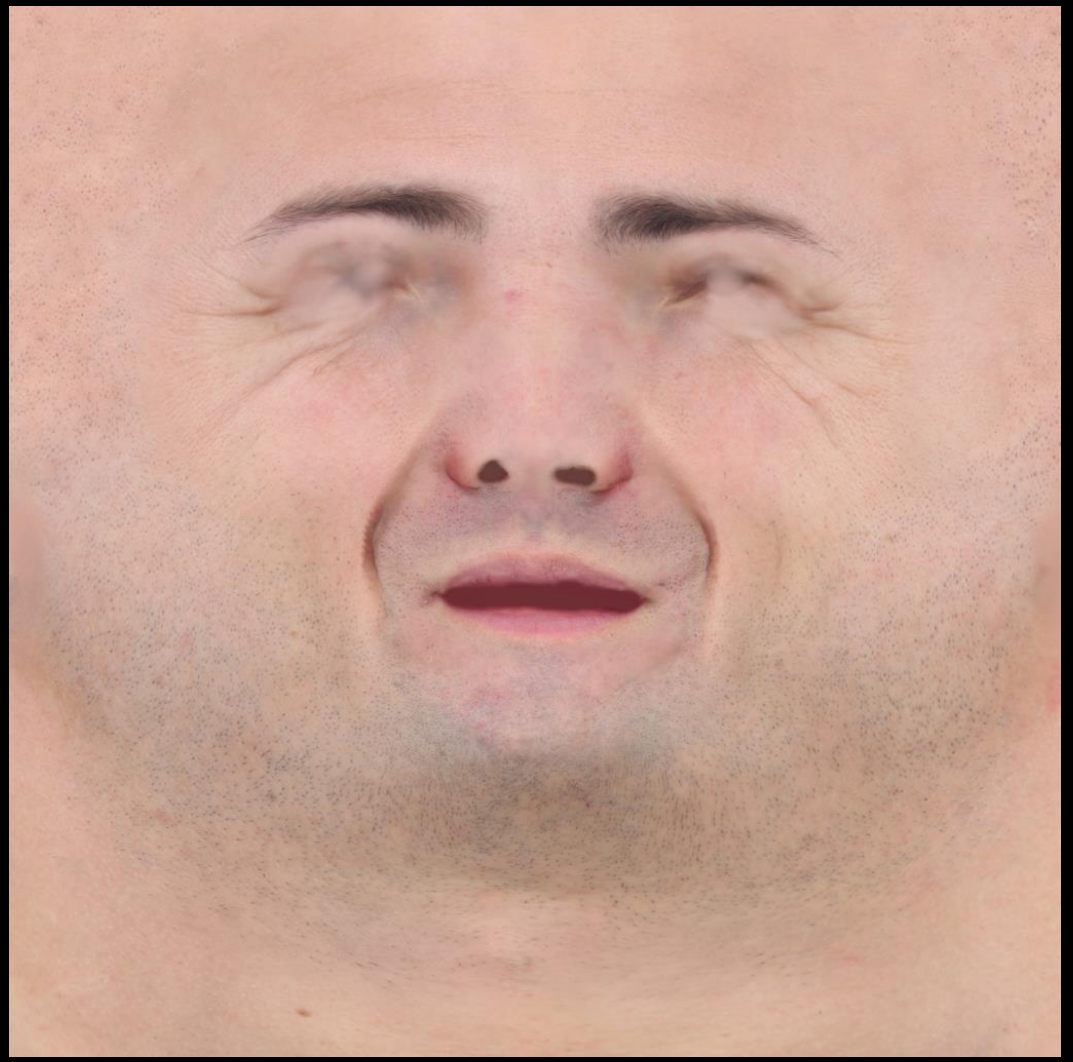
Diffuse Textures:



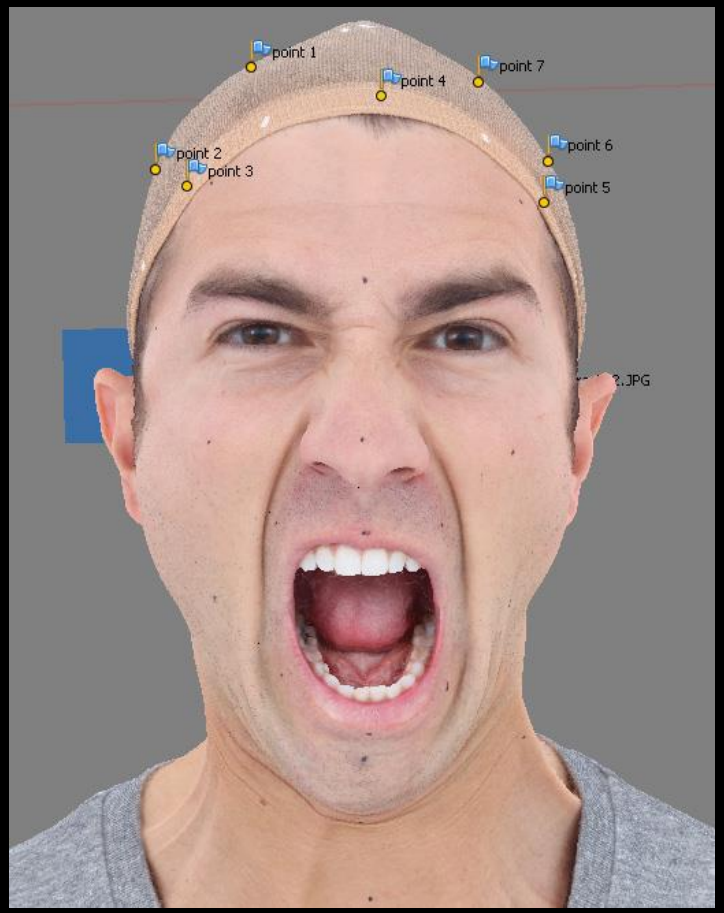


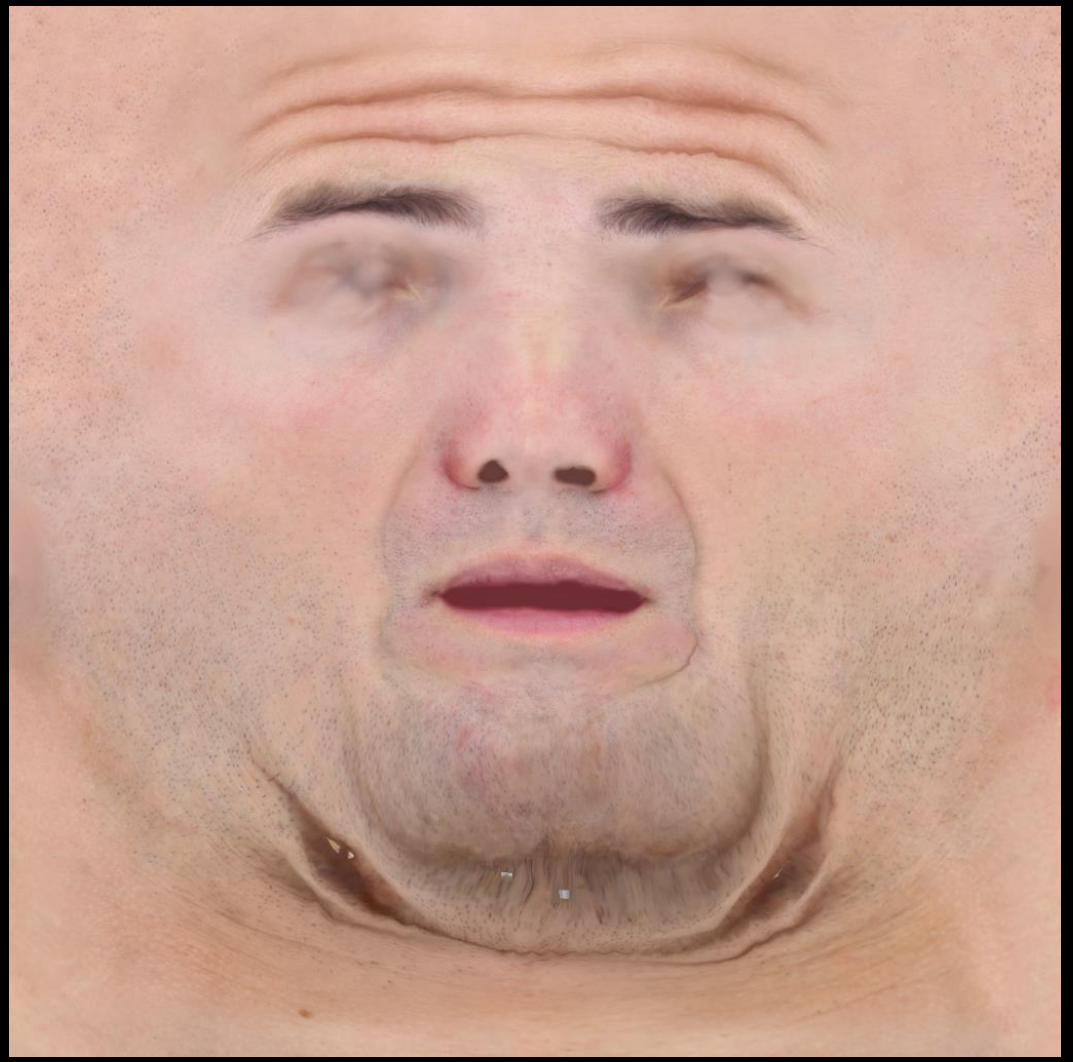
Diffuse Textures:



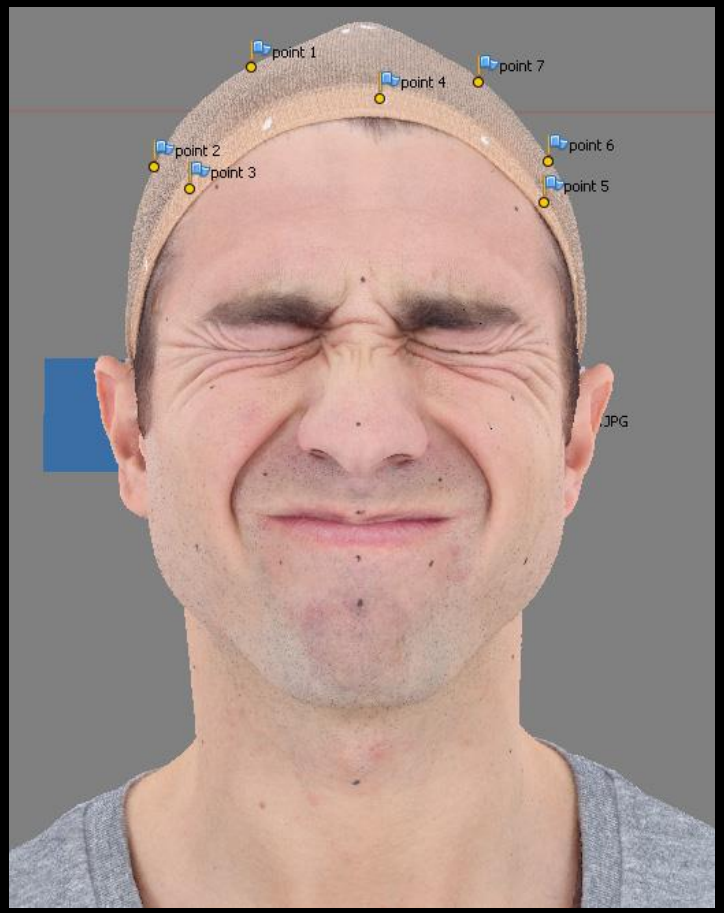


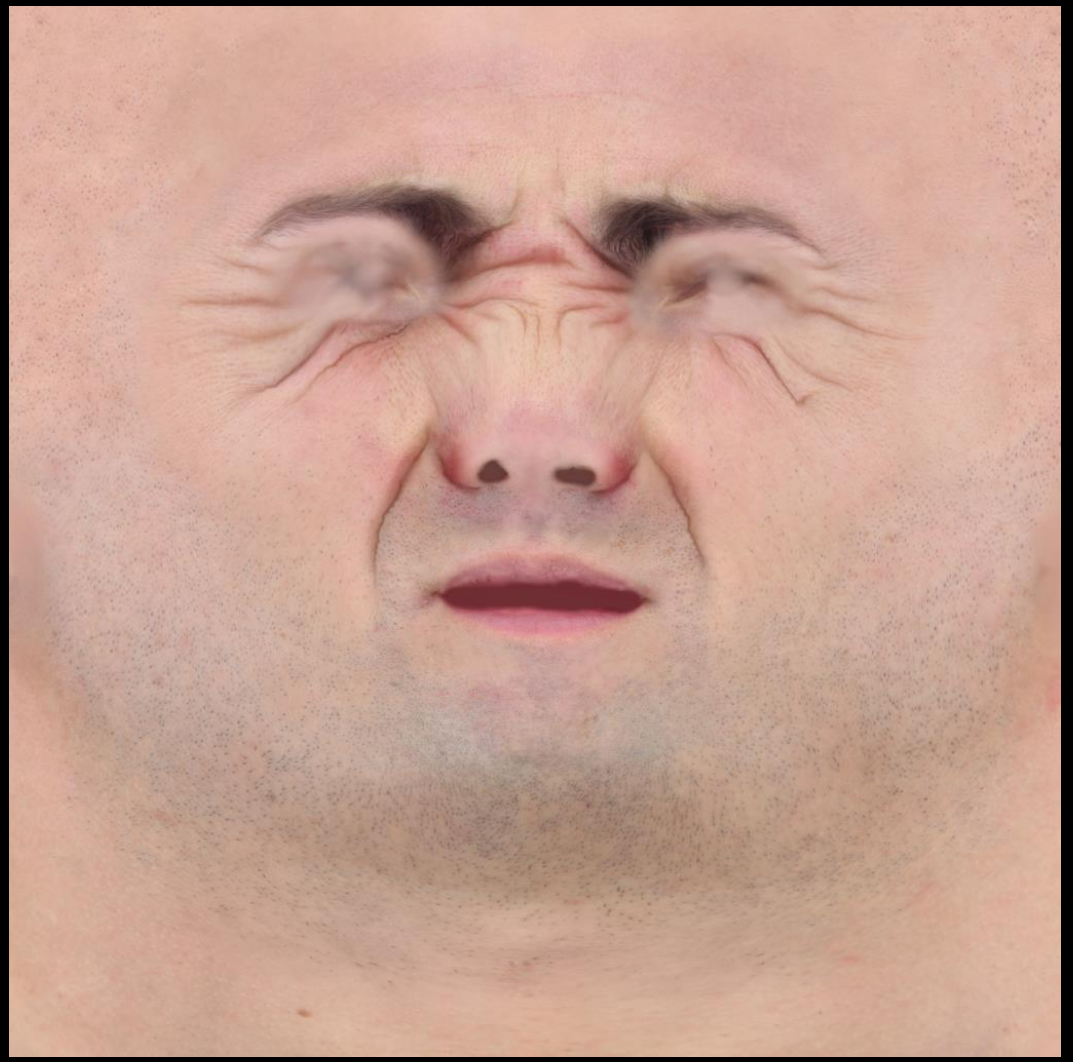
Diffuse Textures:





Diffuse Textures:





Diffuse Textures:

- Diffuse map captures detail from skin stretching
  - It’s not “just wrinkles”
  - Bloodflow is movement too
- Everyone focuses on SPATIAL resolution
  - We need more TEMPORAL resolution

How do we create a rig that:

1. Maintains the geometry of the scans
2. Maintains the diffuse color of the scans
3. Is driven by standard mocap

Background:

1. Georgia Tech (did something)
2. EA WW Vis Group
   - Tiger Woods 2007 (Game and E3 Demo)
   - NFS: Carbon
3. LMNO (Spielberg title at EA)
4. Naughty Dog
5. Contract Work (Microsoft)

Playable Ucap

- GPU Gems 3: Playable Universal Capture
- Tiger Woods 2007 and NFS: Carbon
  - Similar to LA Noire
- Geometry: Nothing fancy
- Diffuse Map: Animates each frame
- Effectively playing a movie on their face

Playable Ucap

- Best facial animation at the time
- Joint Geometry: Deep in uncanny valley
- Joint Geometry + Animated Diffuse: Looks amazing.
- Conclusion: We need animated Diffuse
- Never showed the best data

PTMs:

- Facial Performance Synthesis using Deformation-Driven Polynomial Displacement Maps
- http://gl.ict.usc.edu/Research/PDM/

PTMs:

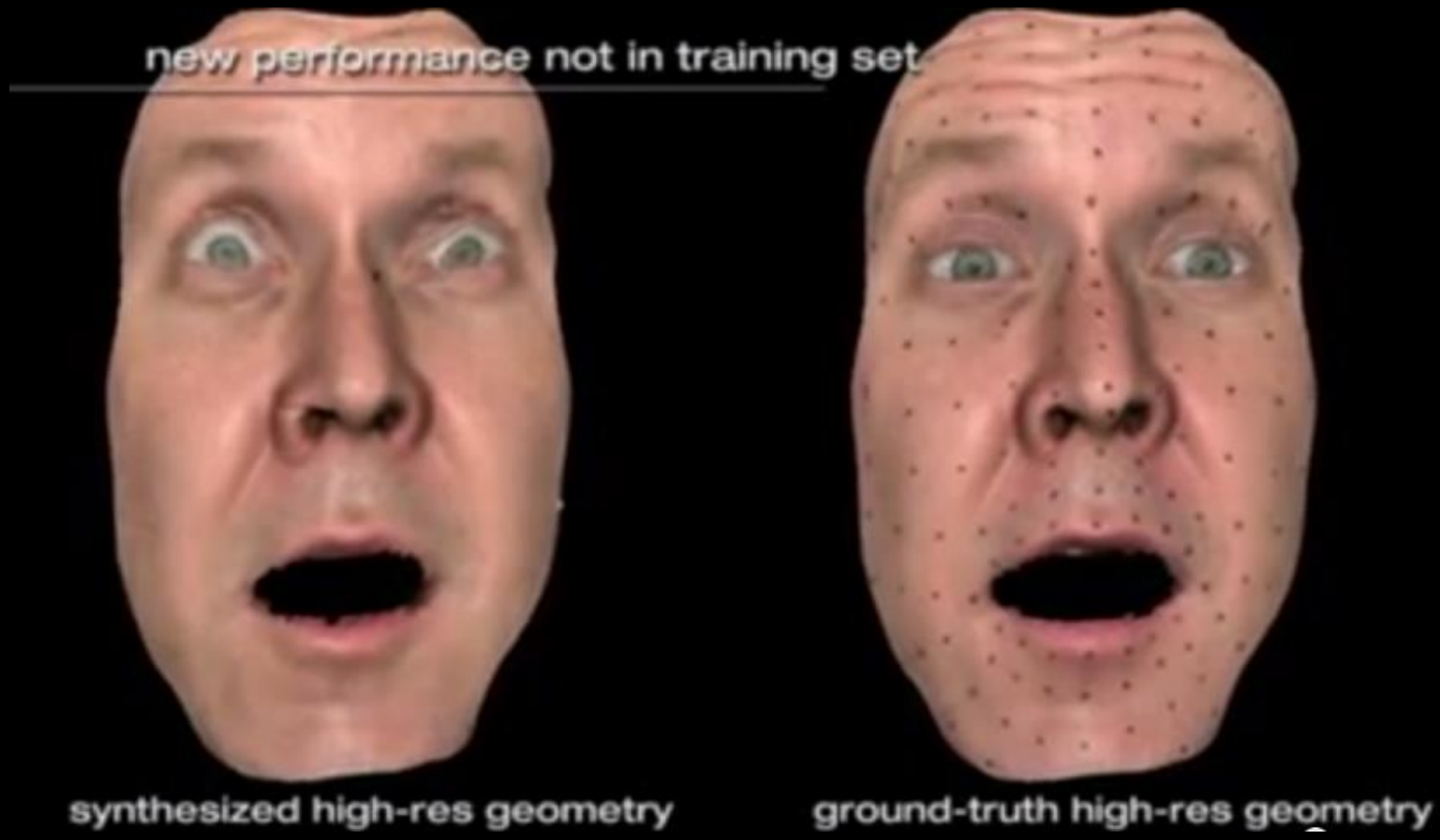
- Facial Performance Synthesis using Deformation-Driven Polynomial Displacement Maps
- http://gl.ict.usc.edu/Research/PDM/

PTMs:

- Facial Performance Synthesis using Deformation-Driven Polynomial Displacement Maps
- http://gl.ict.usc.edu/Research/PDM/

PTMs:

- Facial Performance Synthesis using Deformation-Driven Polynomial Displacement Maps
- http://gl.ict.usc.edu/Research/PDM/



Very Interesting Paper:

- Ucap:
  - Large quantity of high quality data
- PDM Paper:
  - Small quantity of high quality data
    - 6 expressions, about a second each
  - Large quantity of low quality data
  - Use low quality data to drive high quality data

Takeaway: Goal should be…

- Short volume of high quality data
  - Captures subtle face details
- Large quantity of low quality data
  - Markers

Recent Advances:

- Leveraging Motion Capture and 3D Scanning for High-fidelity Facial Performance Acquisition (Haoda Huang, Xin Tong, Hsaing-Tao Wu)
  - Capture may FACE poses
  - Replaces short 4D capture
  - faculty.cs.tamu.edu/jchai/projects/face-TOG-2011/Face-final-v11.pdf

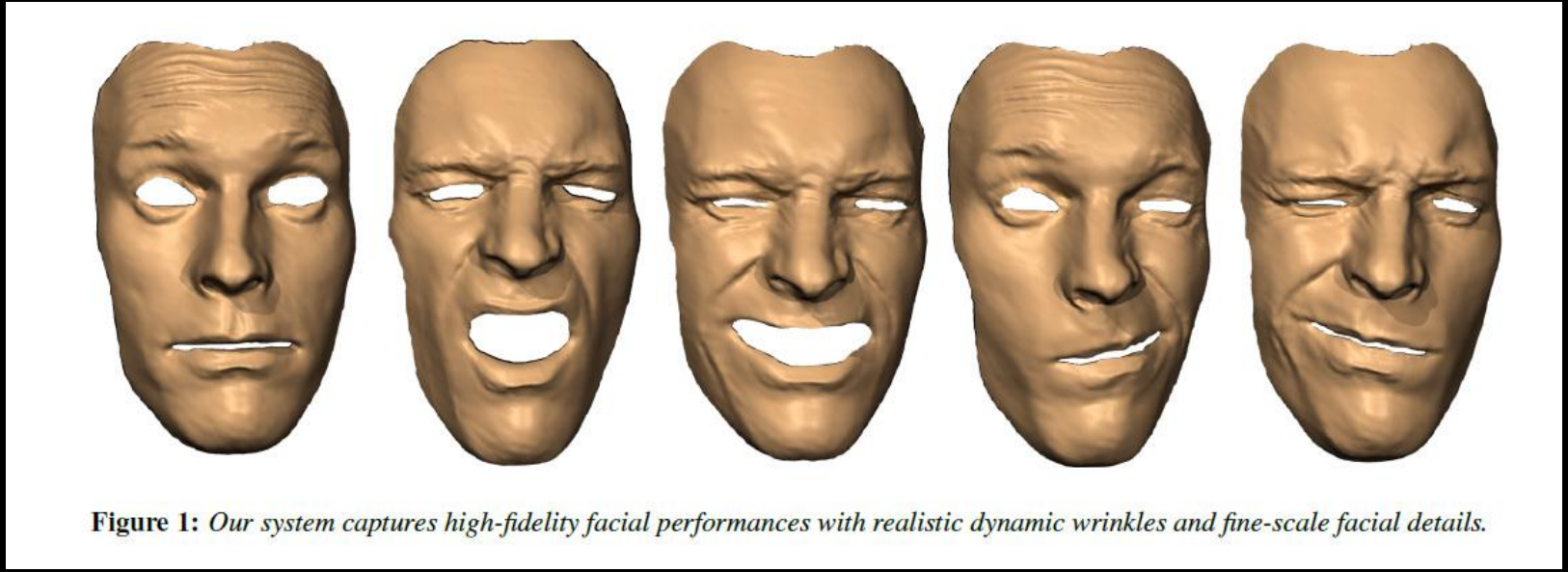
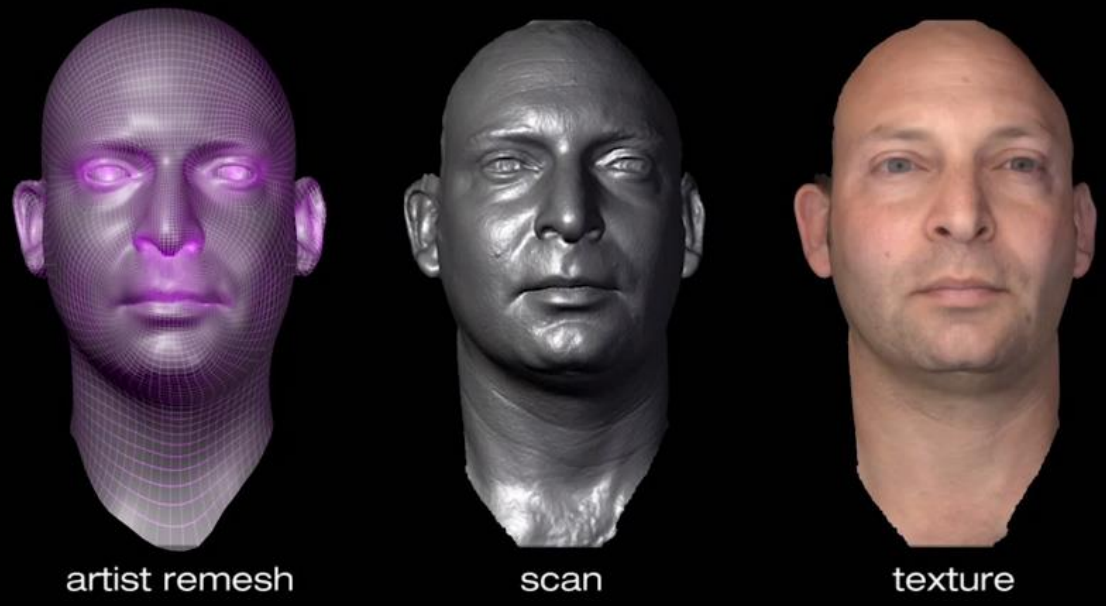


Figure 1: *Our system captures high-fidelity facial performances with realistic dynamic wrinkles and fine-scale facial details.*

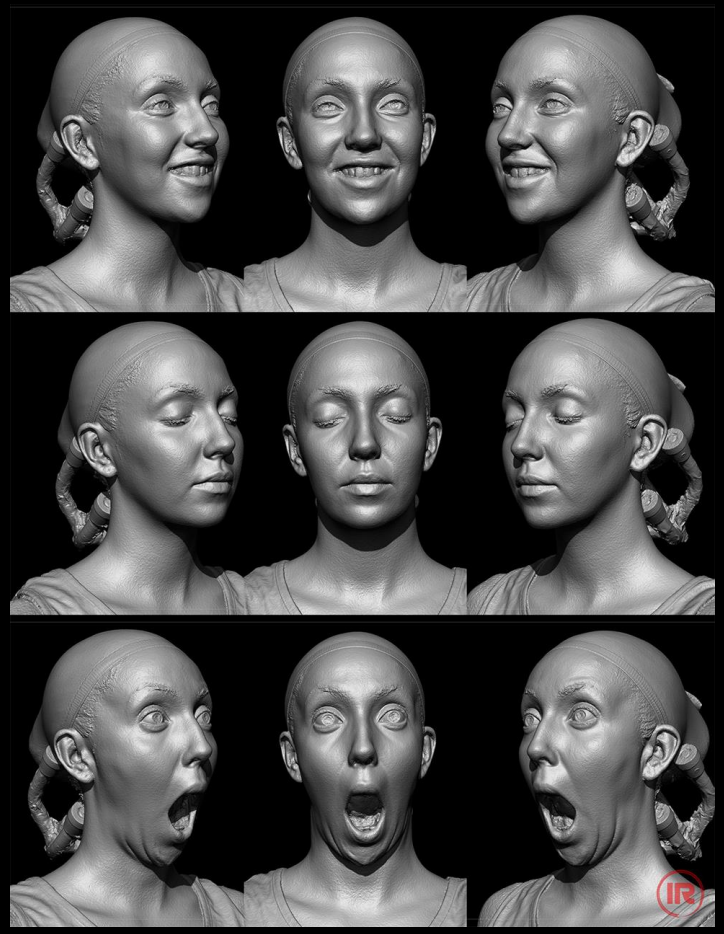
Recent Advances:

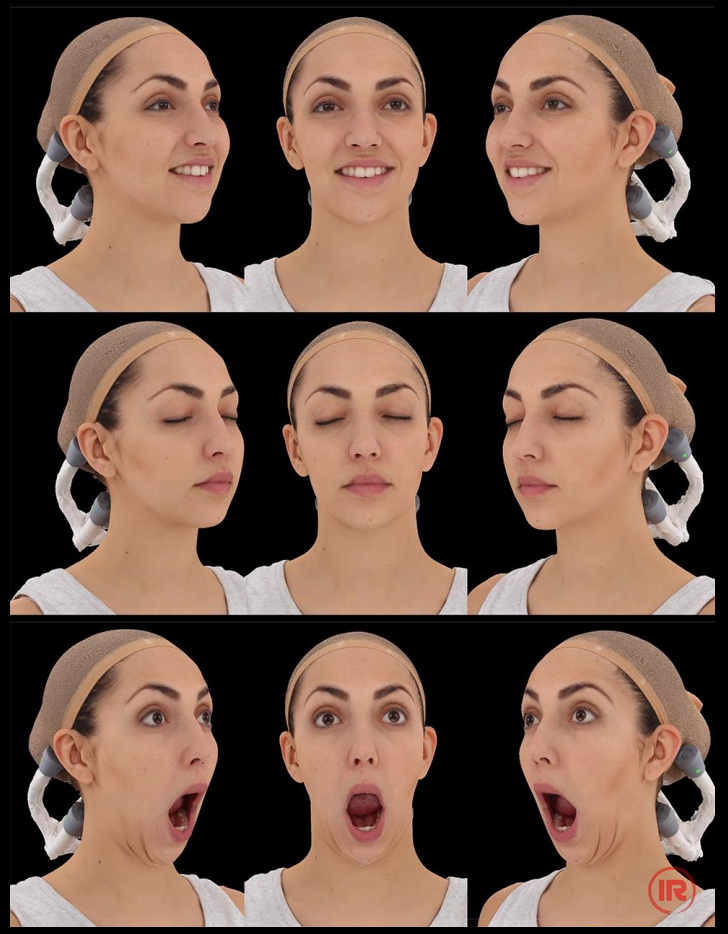
- Vuvuzela (used on Digital Ira)
- Alignment of scans using tracking and optical flow
- http://www.youtube.com/watch?v=IstcFOGwvU4
- Phenomenal alignment



Recent Advances:

- Photogrammetry makes massive improvements (Agisoft Photoscan)
- Infinite-Realities





Workflow Overview:

0) Scan actor.

1) Process raw scans into aligned blendshapes.

2) Compress blendshape textures for realtime playback.

3) Drive blendshapes with mocap

4) Render with skin shading

Step 0: Scan

Solution Used: Photogrammetry

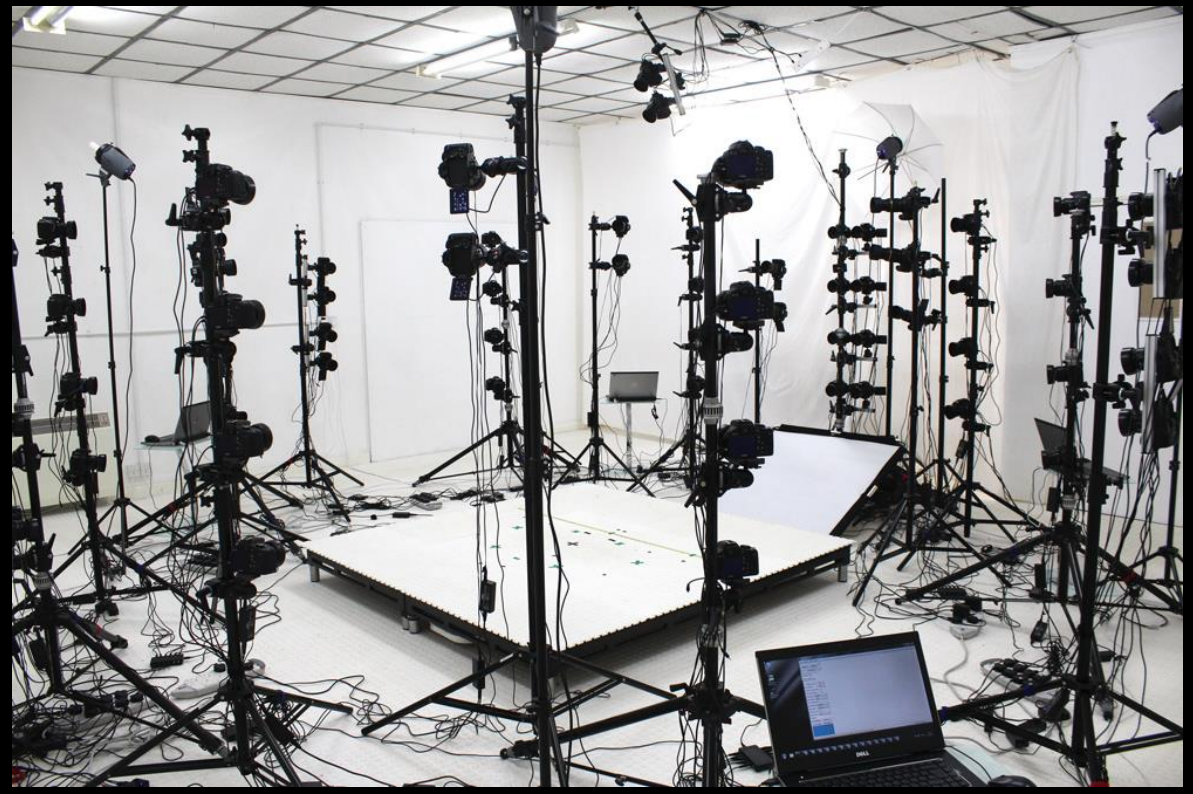
- Infinite Realities with Lee Perry-Smith
- Facial setup with 48 cameras
  - Sync with light
- Also has 150 camera setup for full body
  - (and growing)
- You've probably seen him before…

Photogrammetry Pros:

- Primary Advantage: Instant Capture
  - Sync with light (flash)
  - No alignment fixup
- Can capture fine wrinkles (but not pores)
- Only manual cleanup is ears/mouth

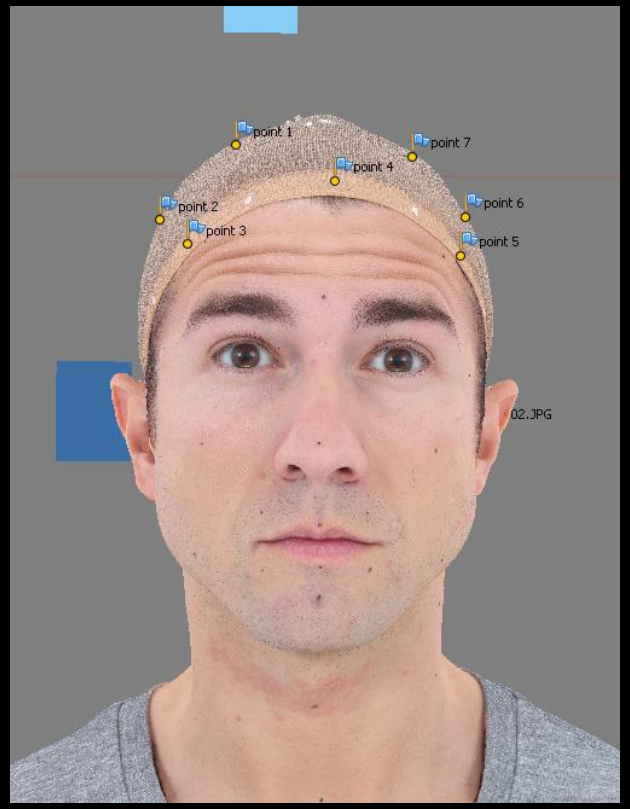
Photogrammetry Cons:

- No Fine Normals
  - You will have to sculpt or extract your normals from the diffuse
- Some lighting baked in
  - Fresnel is an issue



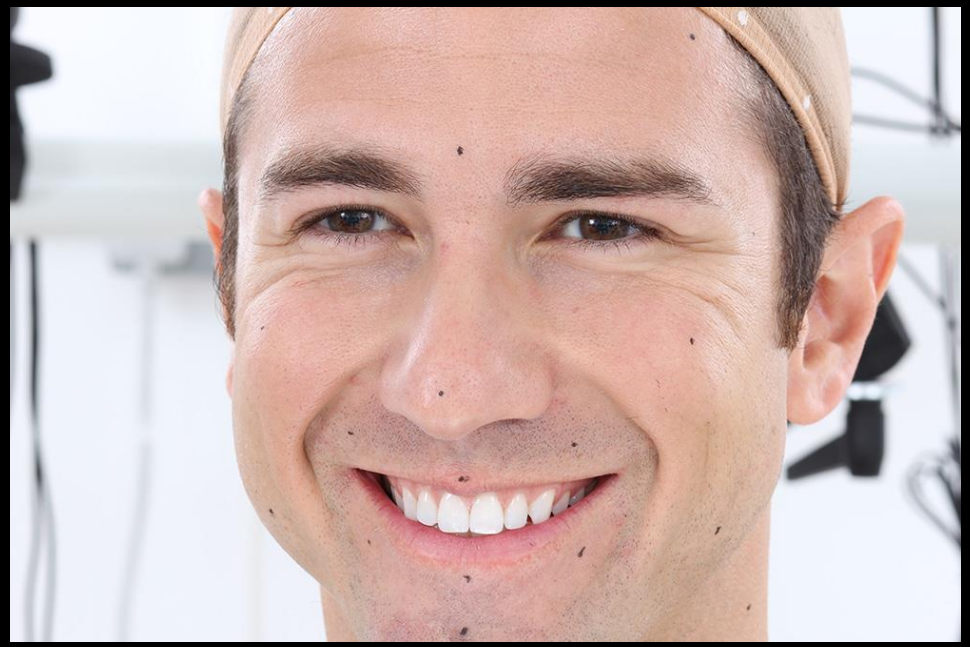
Infinite Reality Scans:

- In this case, 70 expressions with highest quality solve in Photoscan
  - 7.5m polys



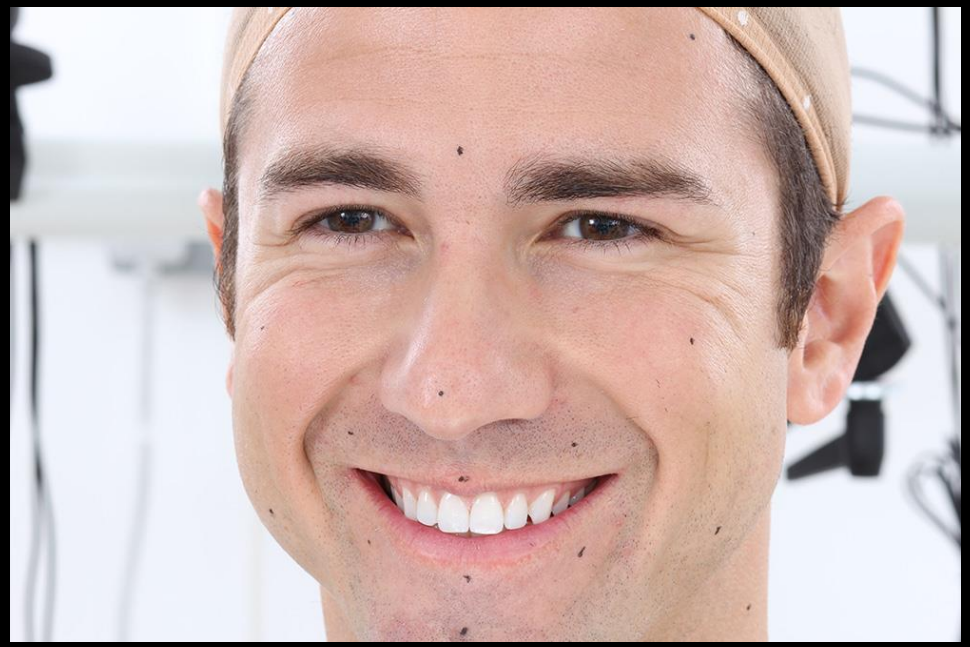
Infinite Reality Scans:

- Great resolution
  - 48 cameras with redundant coverage
  - Each camera is 5184x3456



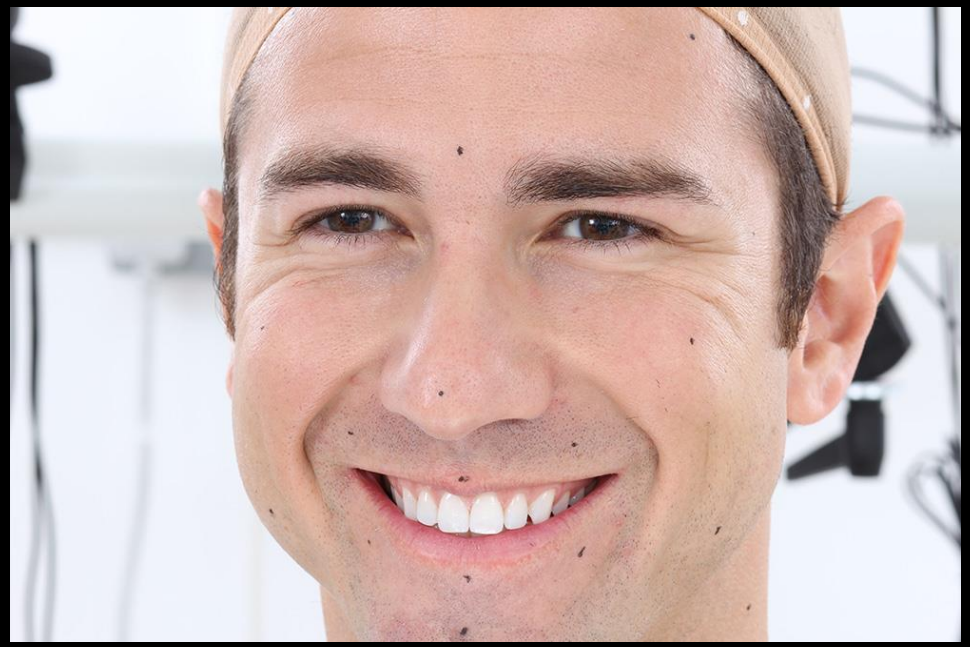
Infinite Reality Scans:

- Great resolution
  - 48 cameras with redundant coverage
  - Each camera is 5184x3456



Infinite-Realities Scans:

- Great resolution
  - 48 cameras with redundant coverage
  - Each camera is 5184x3456



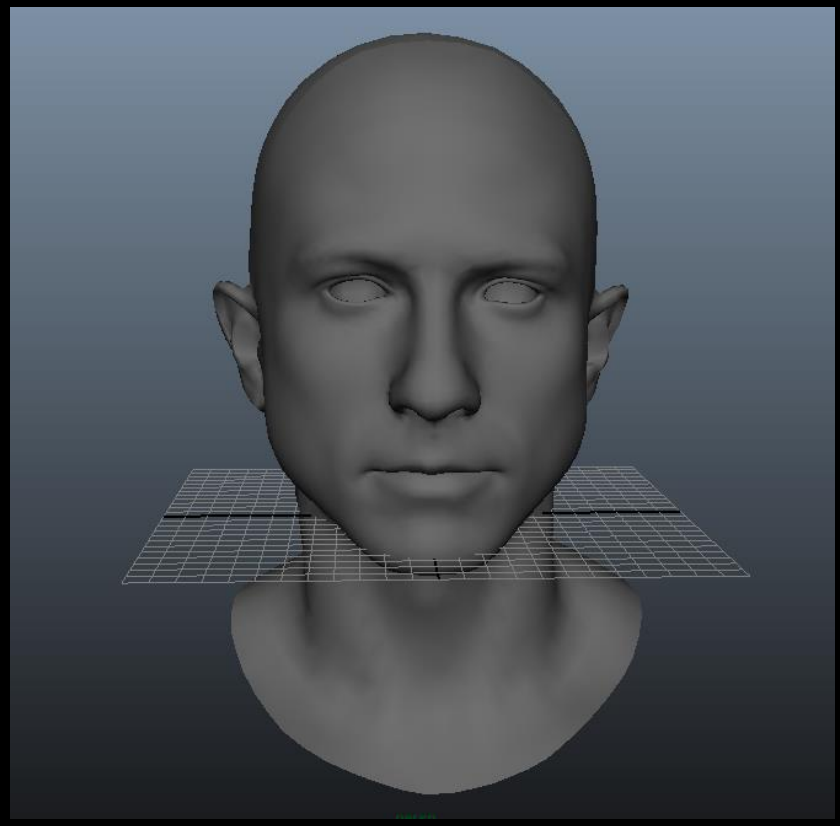
Part 1: Blendshapes

Alignment Overview

1. Place locators at dots
2. Wrap mesh to scan
    1. Move joints to dots
    2. Match to closest point on scan
    3. Relax
    4. Repeat
3. Export head
4. Project textures
    1. Iteratively apply optical flow
        - Curvature and High-Frequency Diffuse
    2. Reapply optical flow results to mesh
5. Done

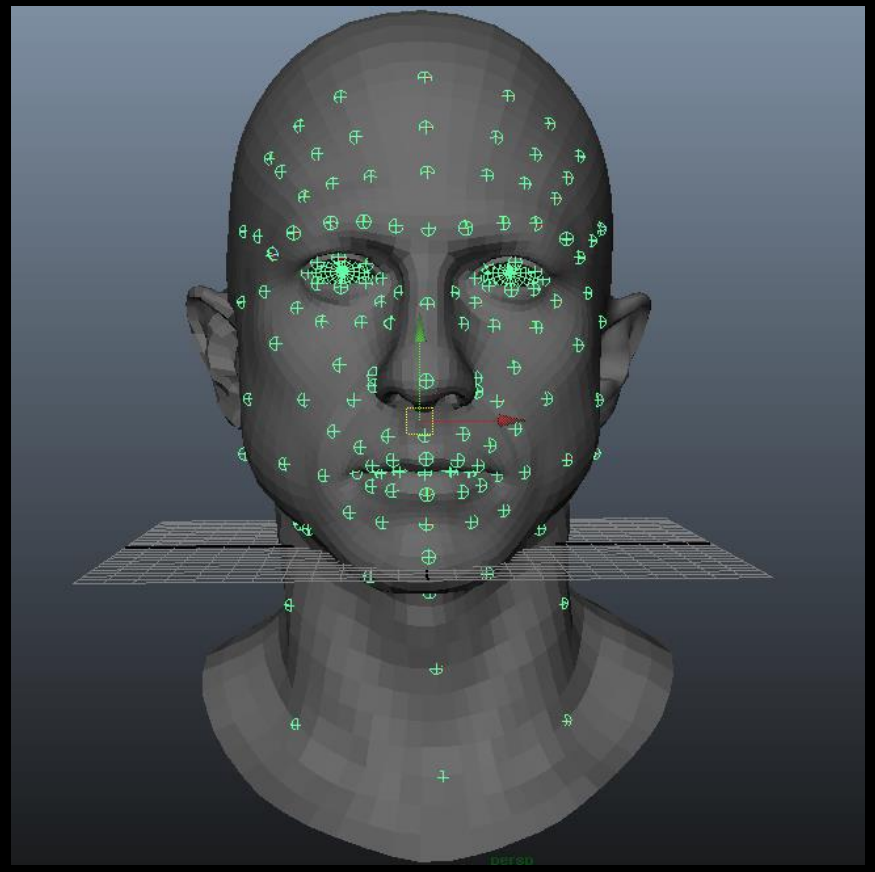
Base Mesh

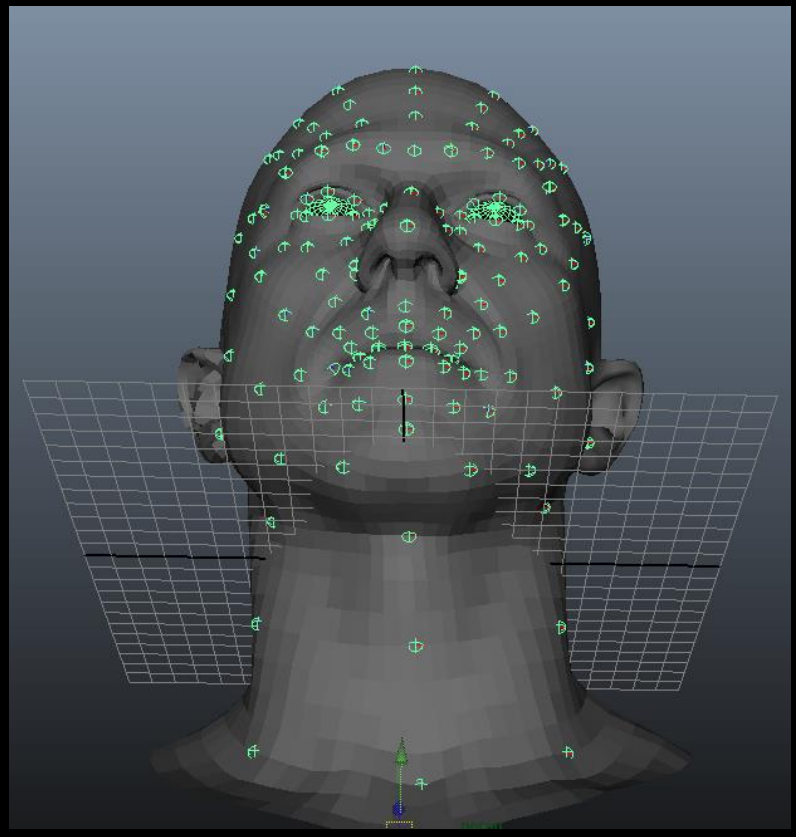
- First task was retopo.
- 4801 verts
- 4684 polys (9368 tris)
- High res, not crazy



Joint Rig

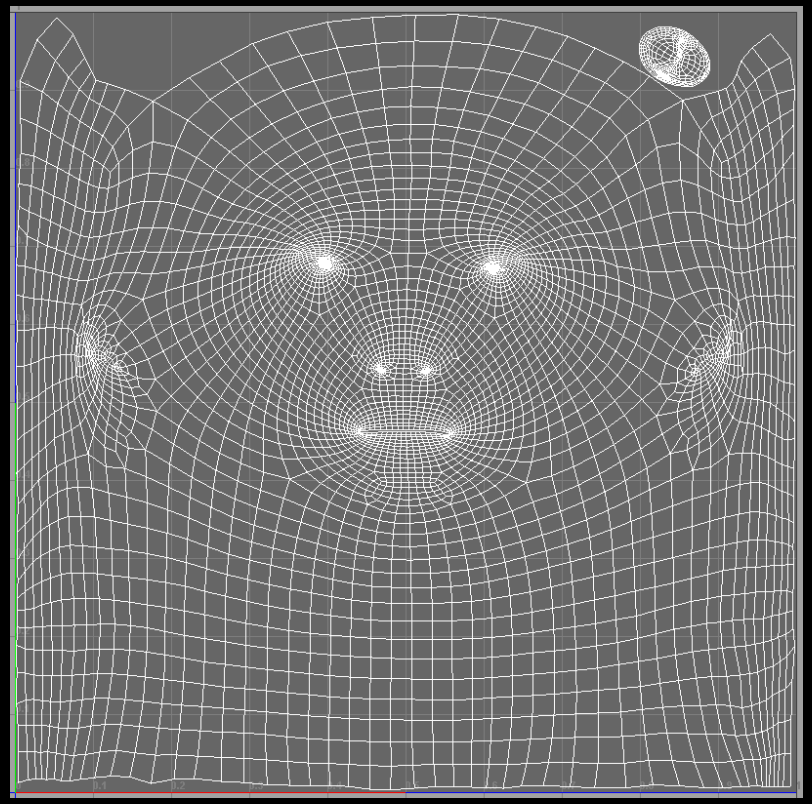
- About 150 joints
- Extra joints on the neck and inner lips





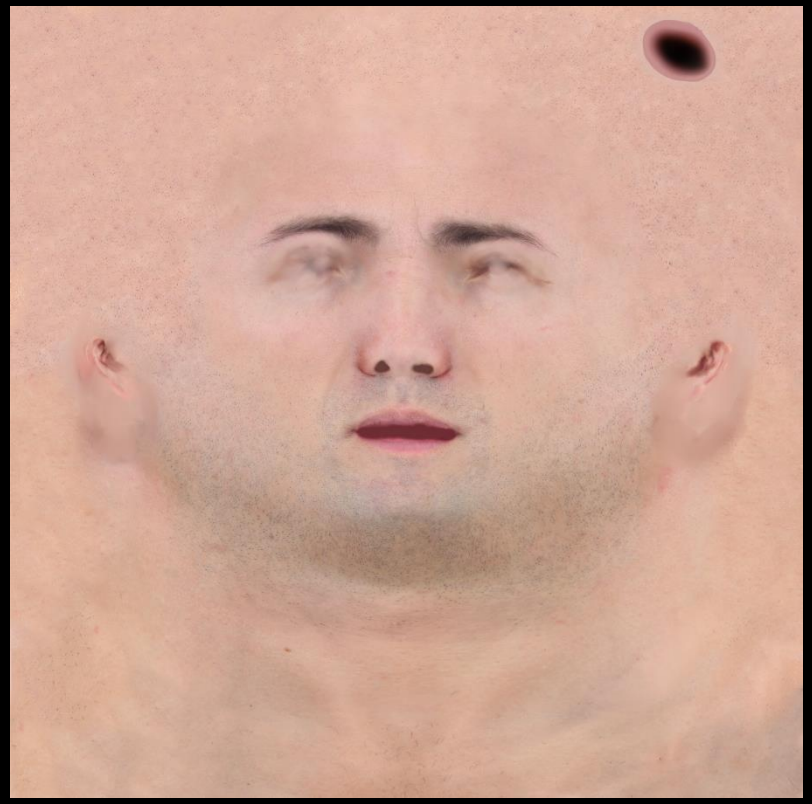
Old-School UVs

- Flattened UVs, optimized for face.
- Replace the middle with animated textures.
  - Animated texture has good layout and minimal stretching
  - Back of head has major stretching, but we don’t care



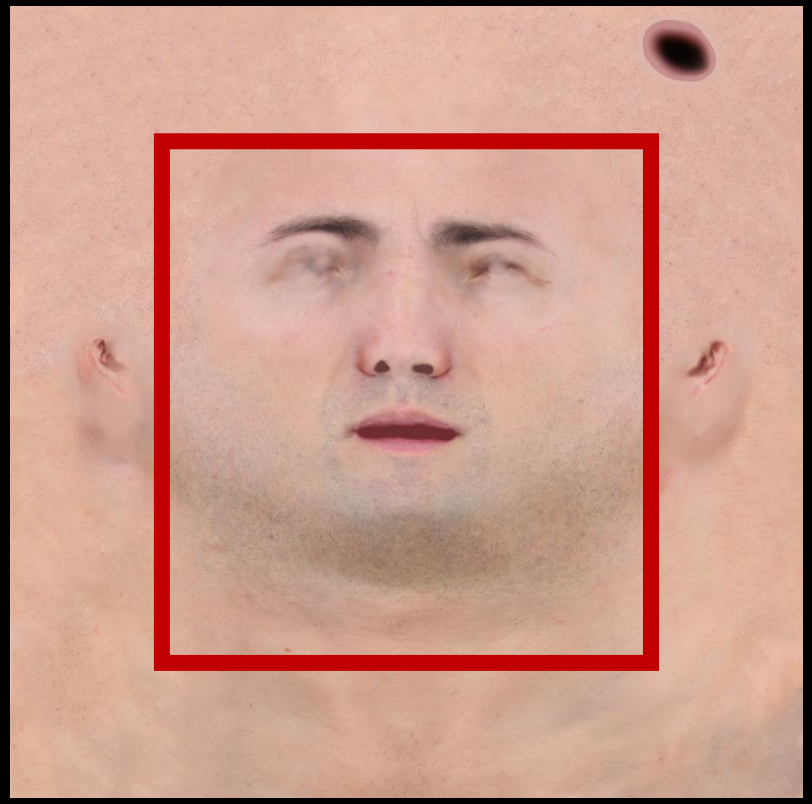
Old-School UVs

- Flattened UVs, optimized for face.
- Replace the middle with animated textures.
  - Animated texture has good layout and minimal stretching
  - Back of head has major stretching, but we don’t care



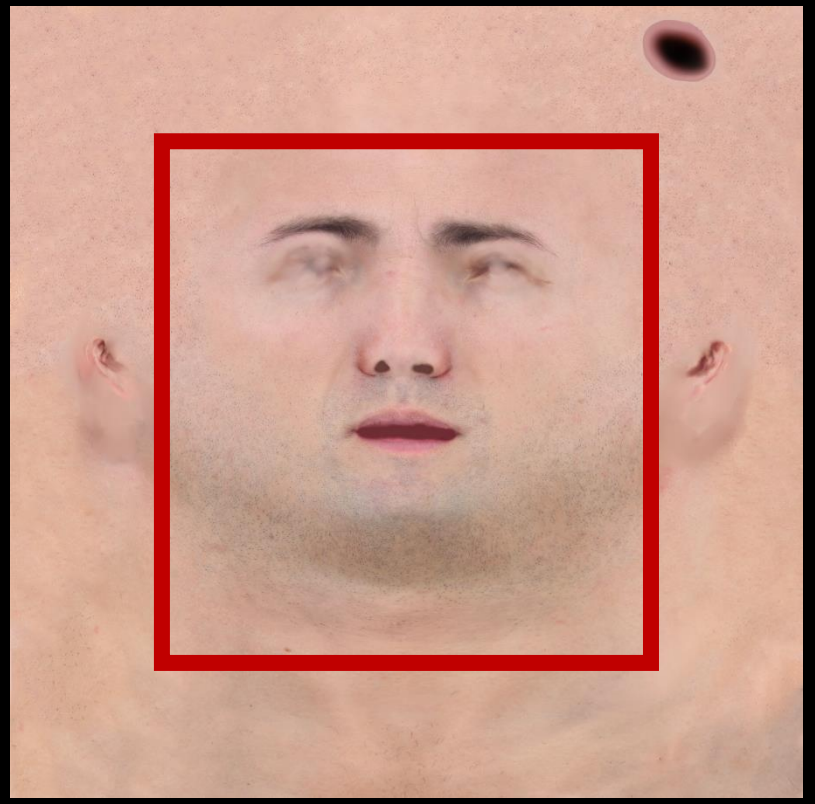
Old-School UVs

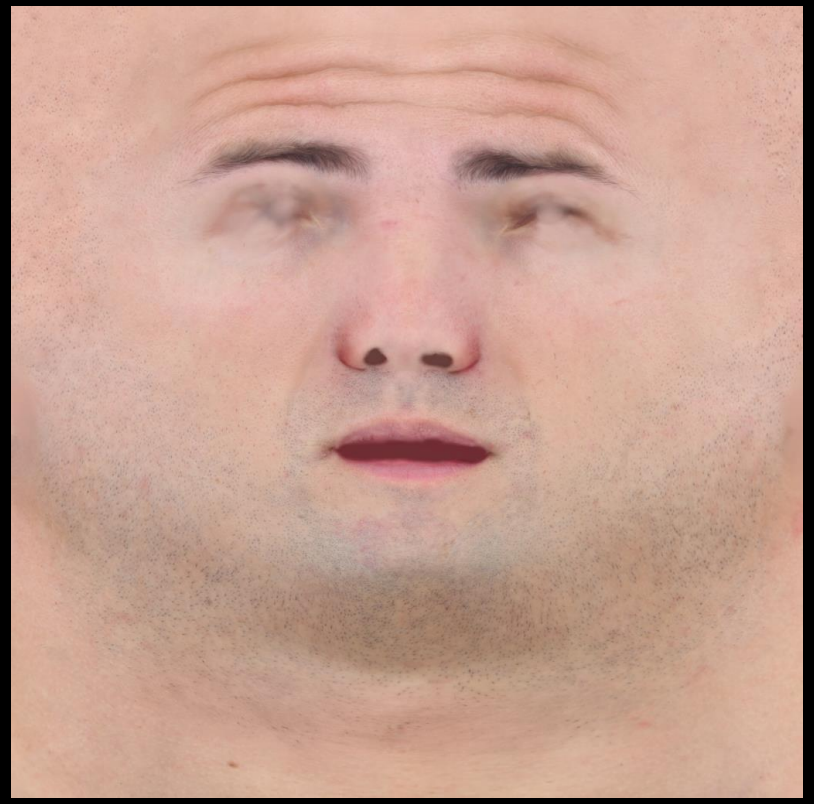
- Flattened UVs, optimized for face.
- Replace the middle with animated textures.
  - Animated texture has good layout and minimal stretching
  - Back of head has major stretching, but we don’t care



Old-School UVs

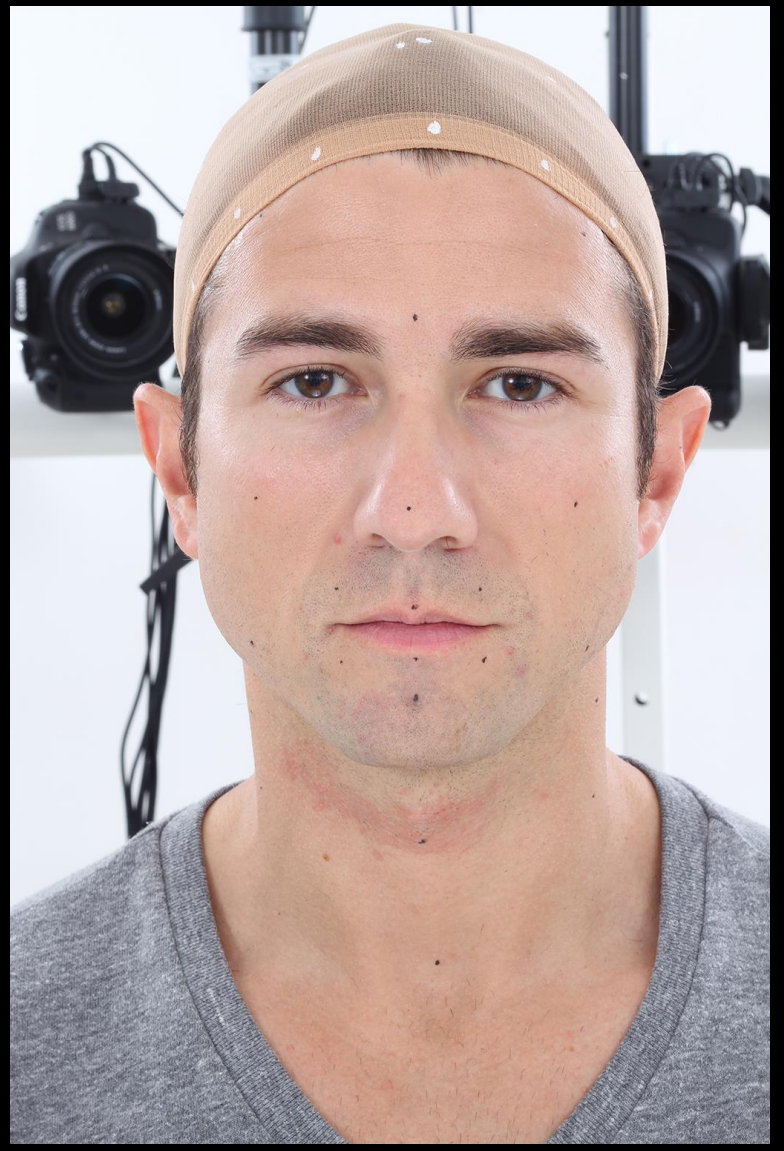
- Flattened UVs, optimized for face.
- Replace the middle with animated textures.
  - Animated texture has good layout and minimal stretching
  - Back of head has major stretching, but we don’t care





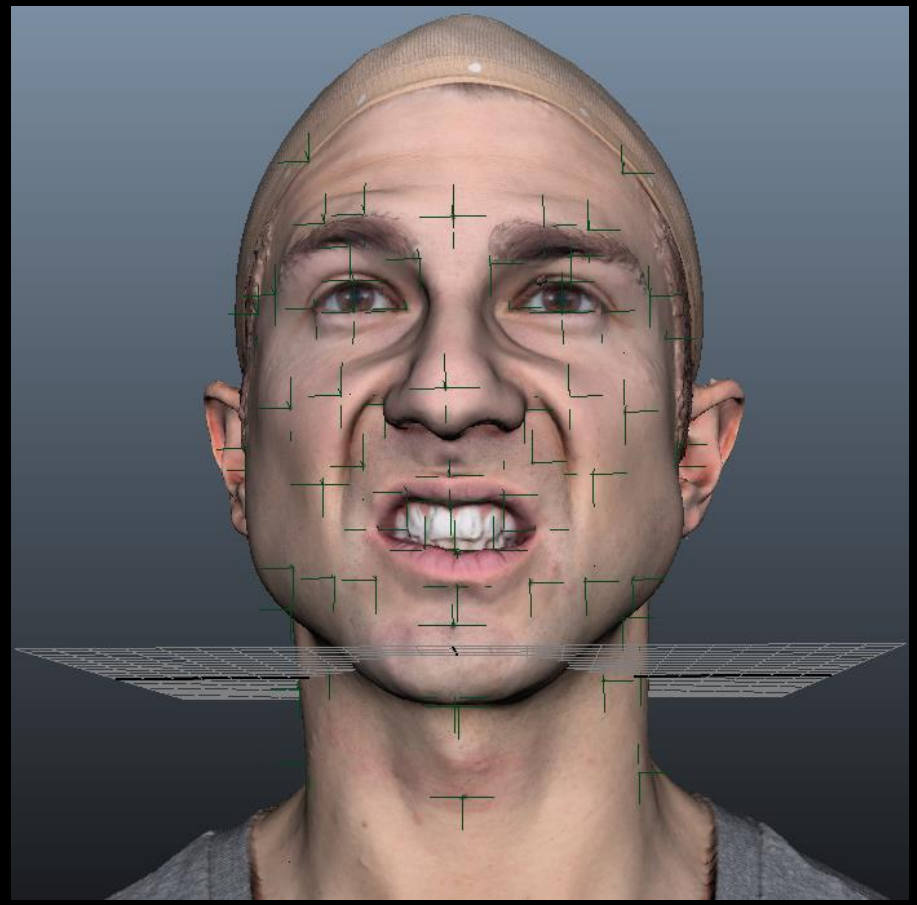
Dots

- You can track without markers
  - HP Duiker convinced me to put some dots on, and I owe him big time.
- My main regret is that I should have had more markers
  - Especially below the jaw!
  - And on the ridge formed by the cheeks.



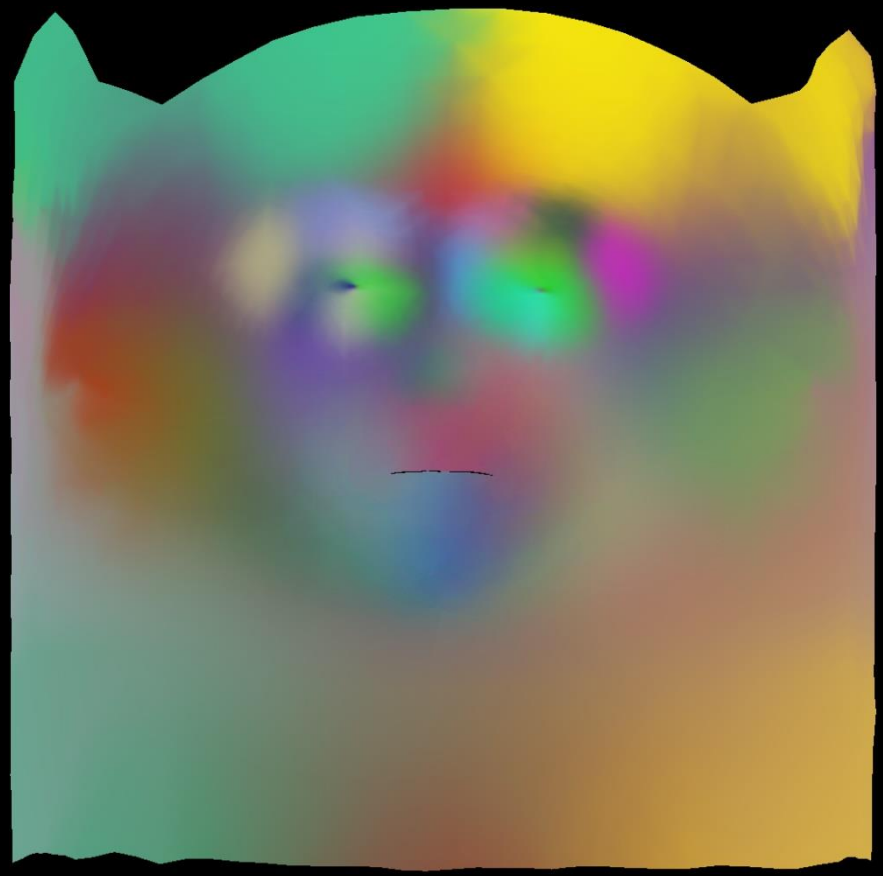
Solving:

- Manually track markers inside Maya
  - Those green crosses are locators
- Was done the most tedious, painful way possible.
  - Placed the locators entirely by hand
  - About 10-15 mins per scan.
    - Spent a few days
  - Could be done much more efficiently



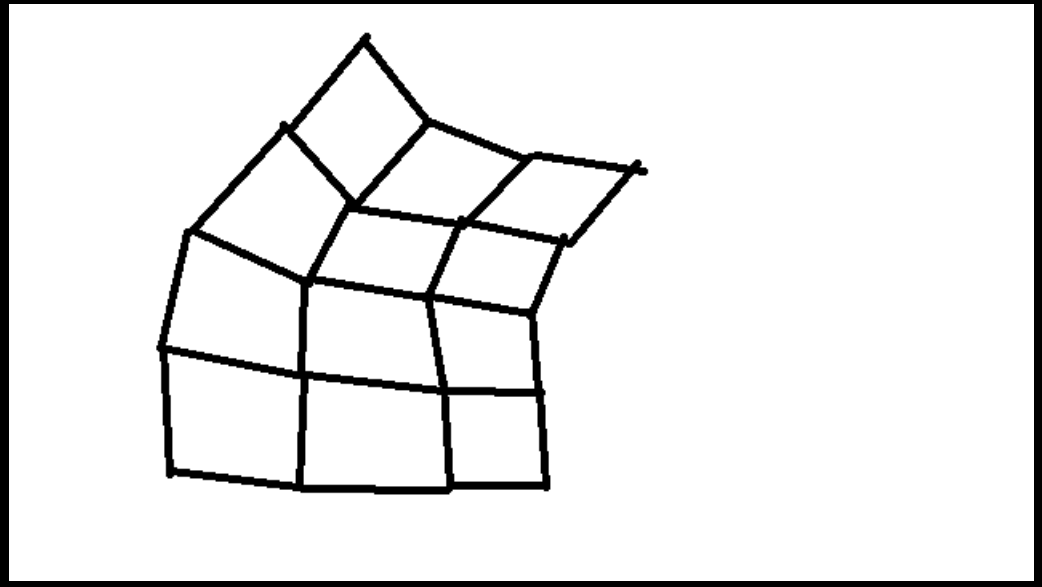
Solving:

- Take the joint in the neutral pose, and find the nearest point on the rig.
- Create rigging automatically.
- Used in iterations.



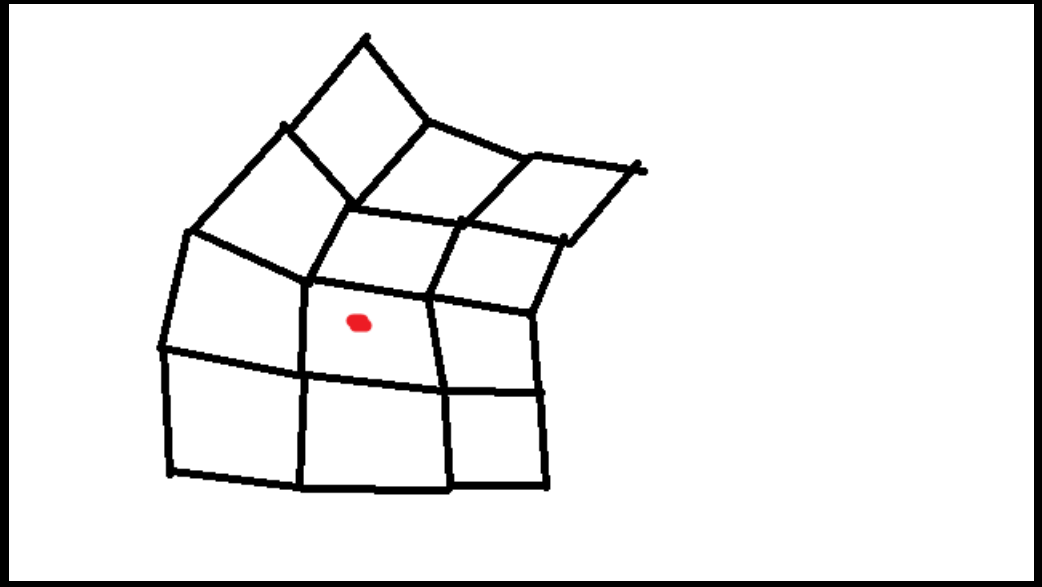
Mesh solving:

- We start with the neutral mesh and want to solve it to a scan.
- Find the locators on the scan.
- Apply the vector translations to the mesh.
- Call this step “SolveForLocators()”.
  - We are tweaking the mesh such that the locators align to the locators on the scan.



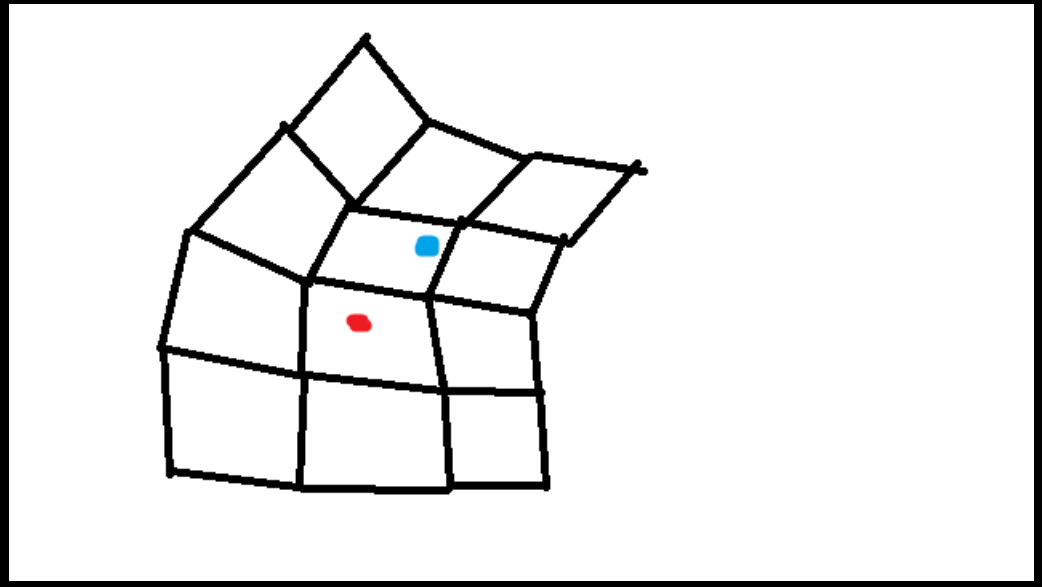
Mesh solving:

- We start with the neutral mesh and want to solve it to a scan.
- Find the locators on the scan.
- Apply the vector translations to the mesh.
- Call this step “SolveForLocators()”.
  - We are tweaking the mesh such that the locators align to the locators on the scan.



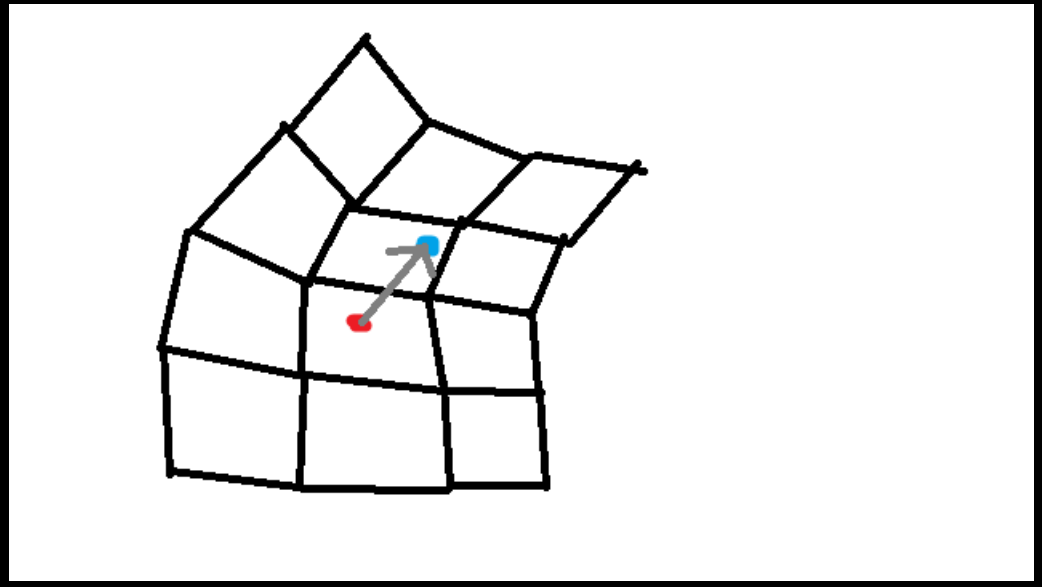
Mesh solving:

- We start with the neutral mesh and want to solve it to a scan.
- Find the locators on the scan.
- Apply the vector translations to the mesh.
- Call this step “SolveForLocators()”.
  - We are tweaking the mesh such that the locators align to the locators on the scan.



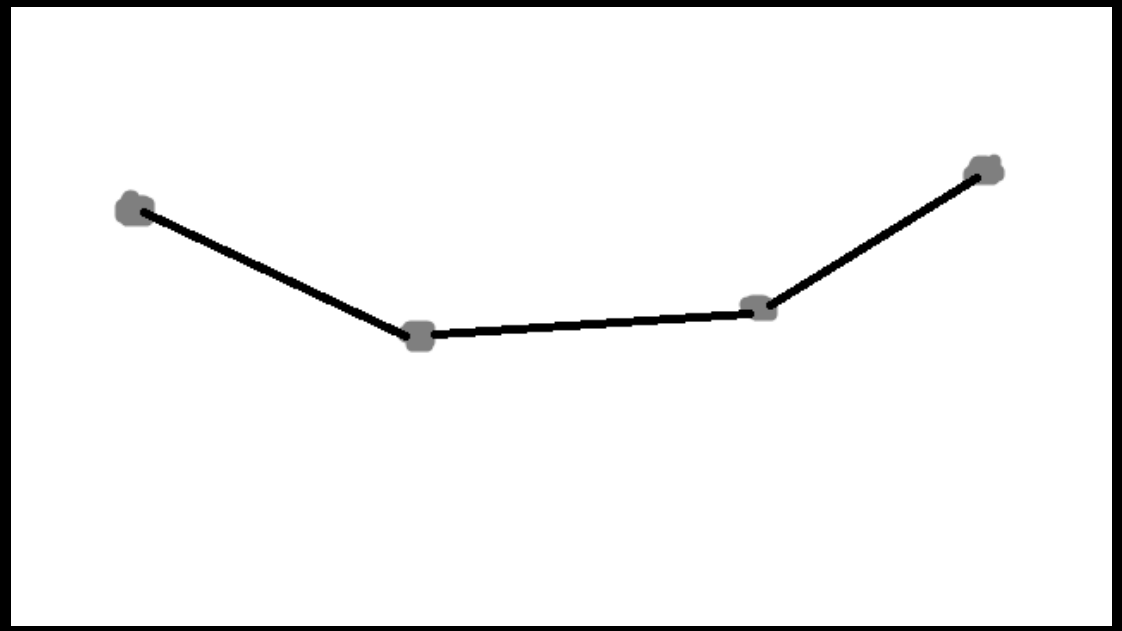
Mesh solving:

- We start with the neutral mesh and want to solve it to a scan.
- Find the locators on the scan.
- Apply the vector translations to the mesh.
- Call this step “SolveForLocators()”.
  - We are tweaking the mesh such that the locators align to the locators on the scan.



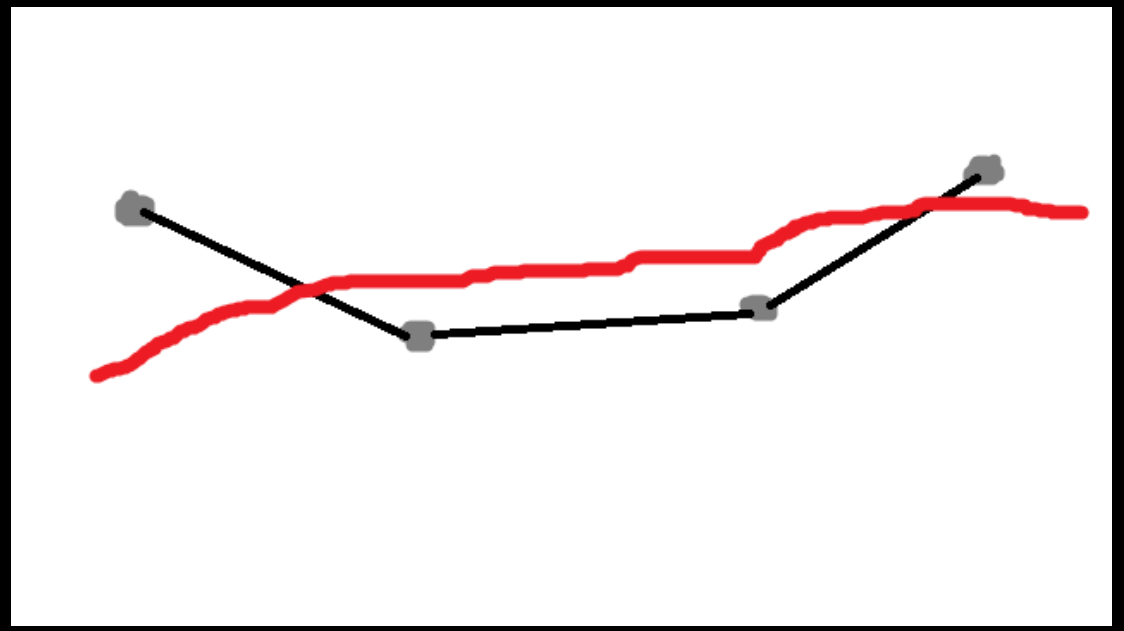
Mesh solving:

- Another operation we can do is lock to the scan.
- For each vertex:
  - Move the vertex to the nearest point on the scan.
- Call this operation “LockToScan()”.



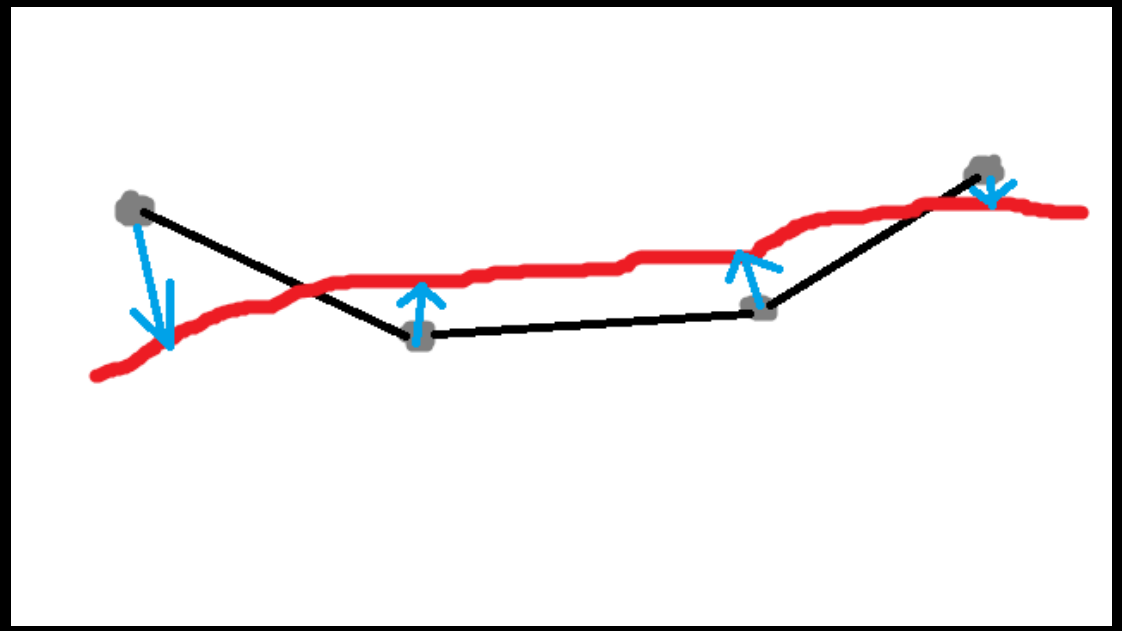
Mesh solving:

- Another operation we can do is lock to the scan.
- For each vertex:
  - Move the vertex to the nearest point on the scan.
- Call this operation “LockToScan()”.



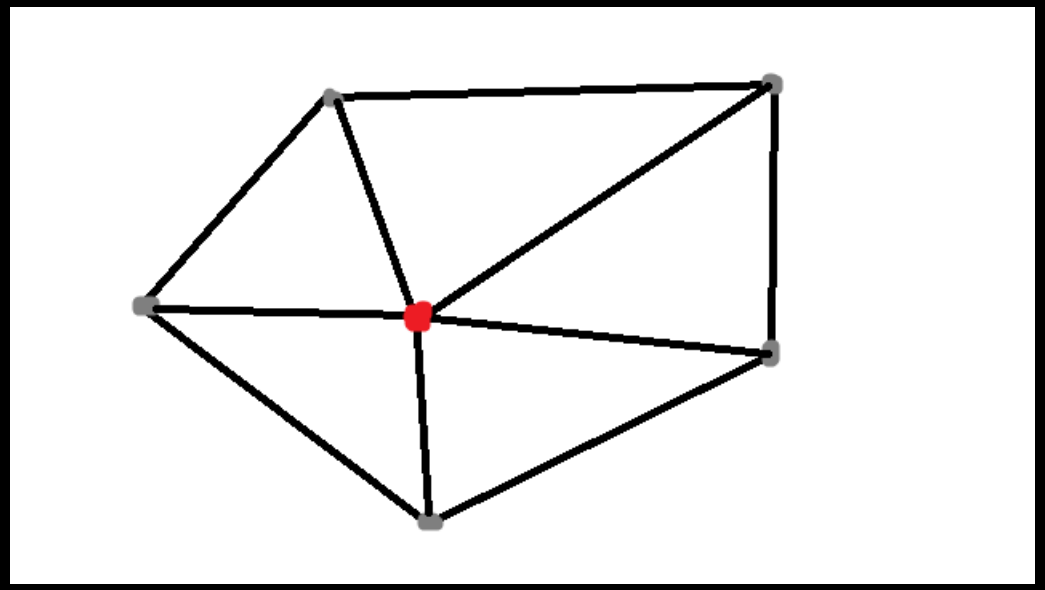
Mesh solving:

- Another operation we can do is lock to the scan.
- For each vertex:
  - Move the vertex to the nearest point on the scan.
- Call this operation “LockToScan()”.



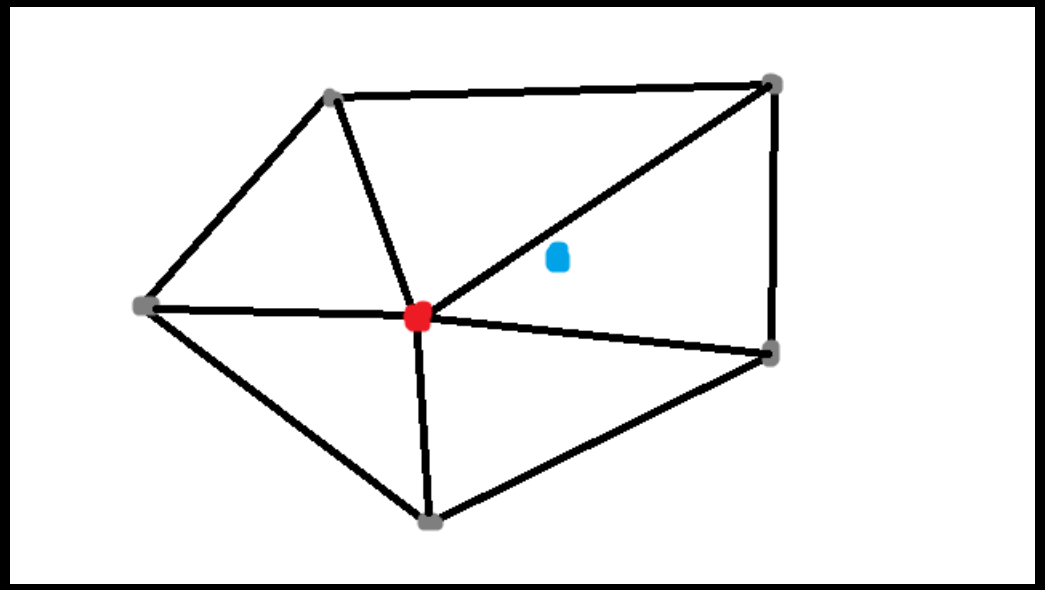
One more operation:

- Laplacian smoothing
- The laplacian is the offset of a vertex from the average of its neighbors.
- Intuitively, we want a solution that minimizes sharp edges.
- Call this operation “RelaxMesh()”



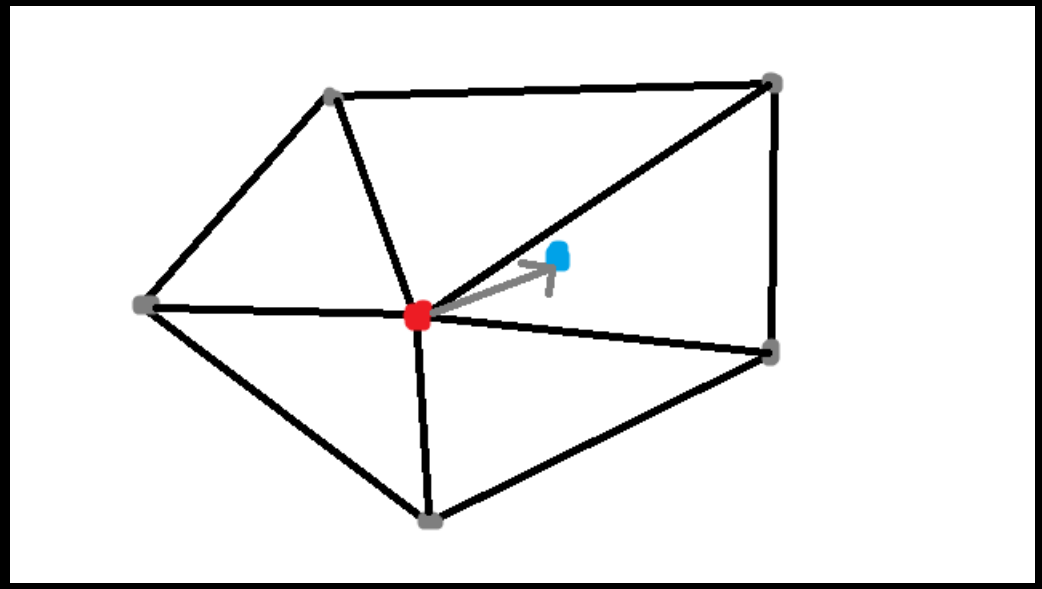
One more operation:

- Laplacian smoothing
- The laplacian is the offset of a vertex from the average of its neighbors.
- Intuitively, we want a solution that minimizes sharp edges.
- Call this operation “RelaxMesh()”



One more operation:

- Laplacian smoothing
- The laplacian is the offset of a vertex from the average of its neighbors.
- Intuitively, we want a solution that minimizes sharp edges.
- Call this operation “RelaxMesh()”

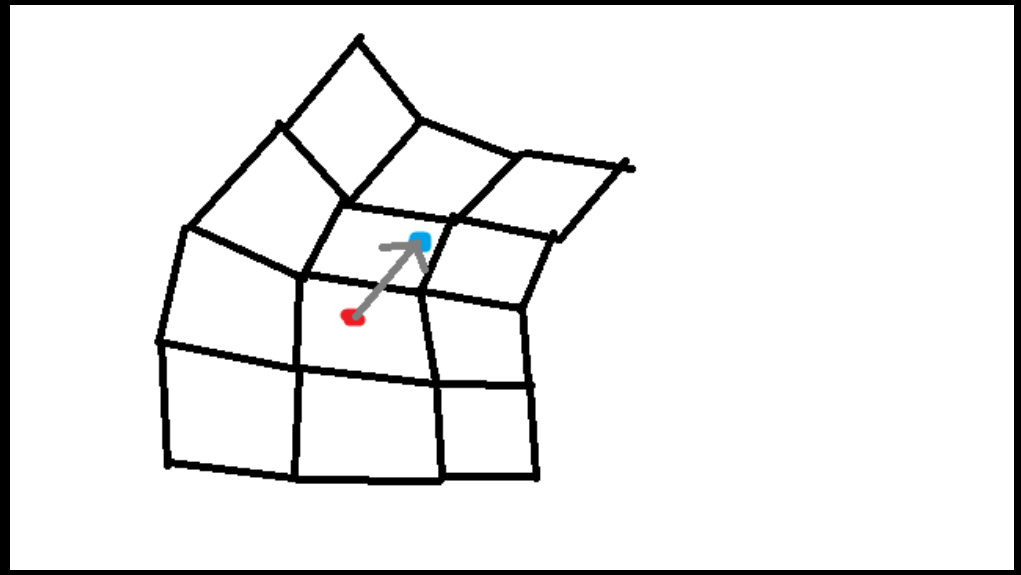


Solving:

For the initial solve, we just have a big for loop.

For a bunch of iterations:

```
SolveForLocators();
LockToScan();
RelaxMesh();
```

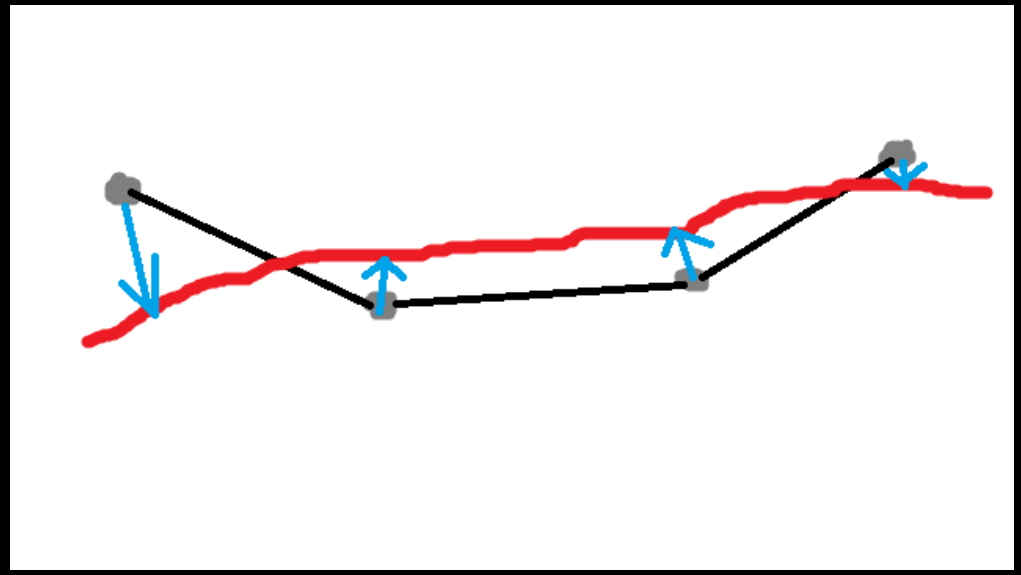


Solving:

For the initial solve, we just have a big for loop.

For a bunch of iterations:

```
SolveForLocators();
LockToScan();
RelaxMesh();
```

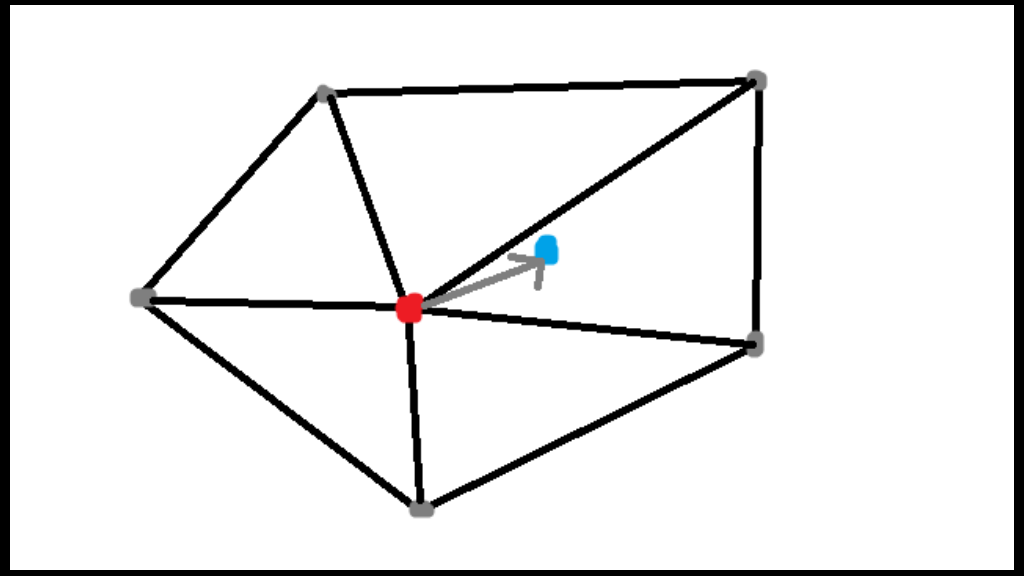


Solving:

For the initial solve, we just have a big for loop.

For a bunch of iterations:

```
SolveForLocators();
LockToScan();
RelaxMesh();
```

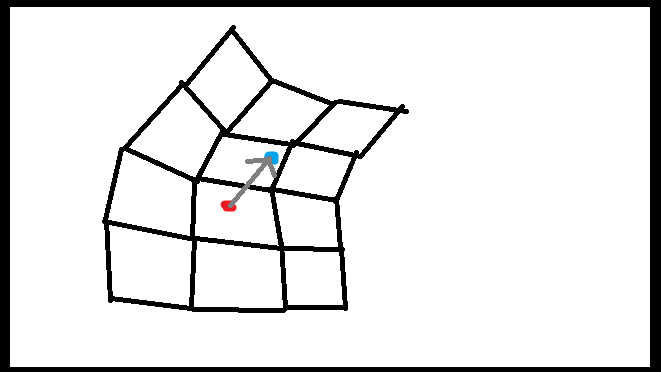


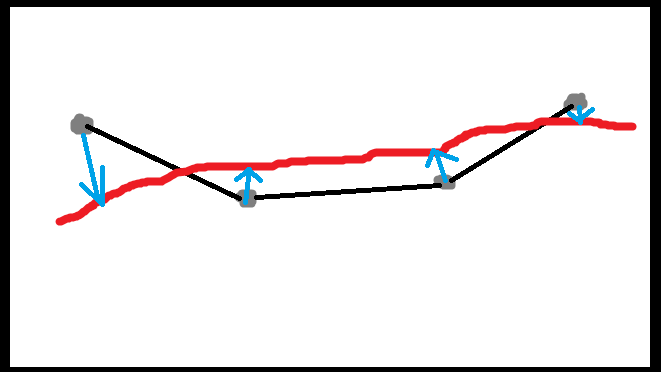
Solving:

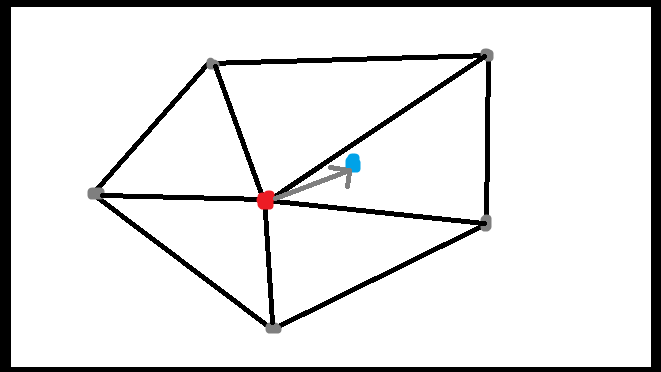
For the initial solve, we just have a big for loop.

For a bunch of iterations:

```
SolveForLocators();
LockToScan();
RelaxMesh();
```

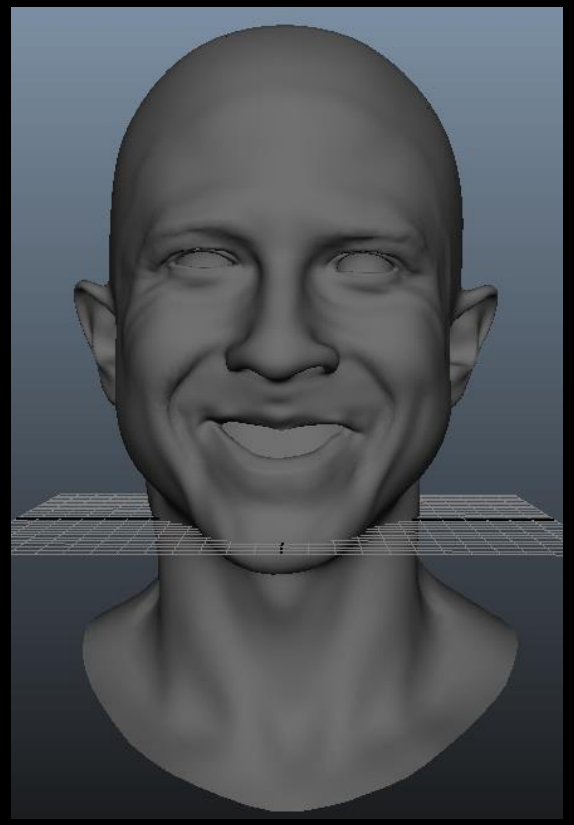






After solving:

- Left is both, middle is the solved rig, right is the scan



Solving Meshes:

- That process solves half of our problem.
- We need to align the rig to the scan:
  - Done
- We need the UVs to perfectly match.
  - This is the hard part.

Project the textures as a starting point.

Project the textures as a starting point.

Project the textures as a starting point.

Project the textures as a starting point.

Project the textures as a starting point.

Project the textures as a starting point.

Let’s take a break and talk about optical flow…

Globally optimize optical flow.

Inspiration:

- Leveraging Motion Capture and 3D Scanning for High-fidelity Facial Performance Acquisition
- http://faculty.cs.tamu.edu/jchai/projects/face-TOG-2011/Face-final-v11.pdf
- Match from optical flow using curvature

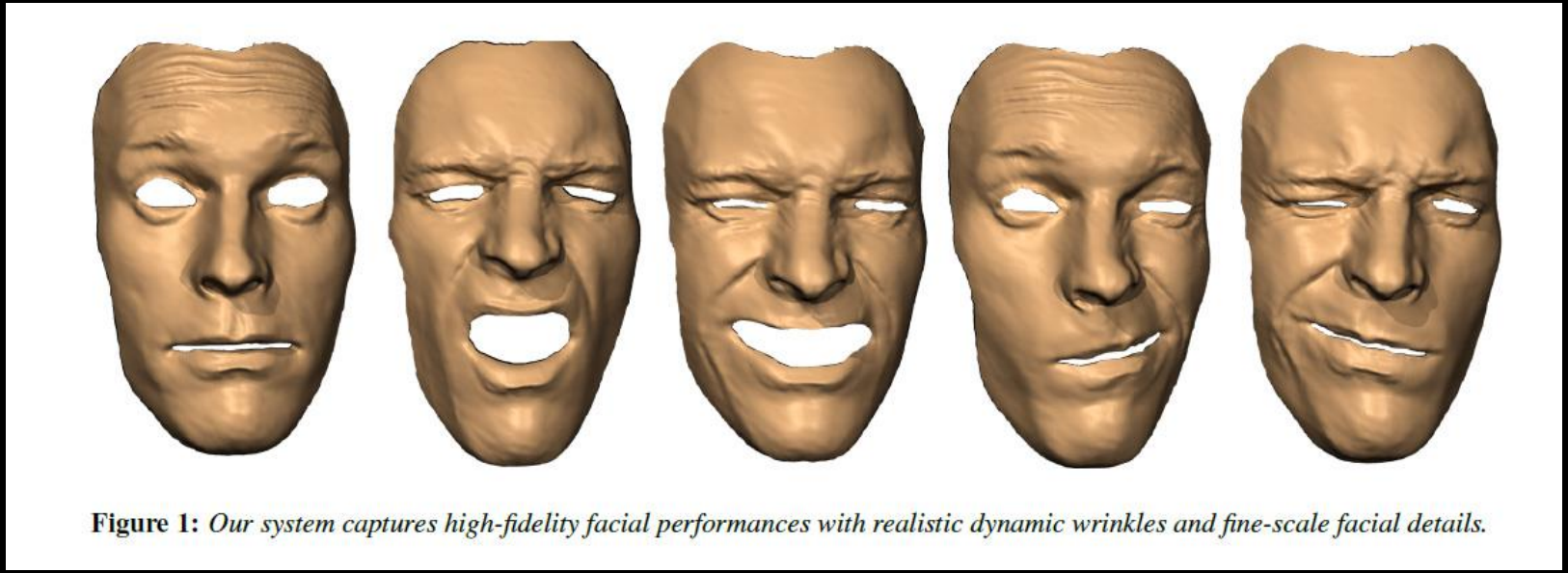
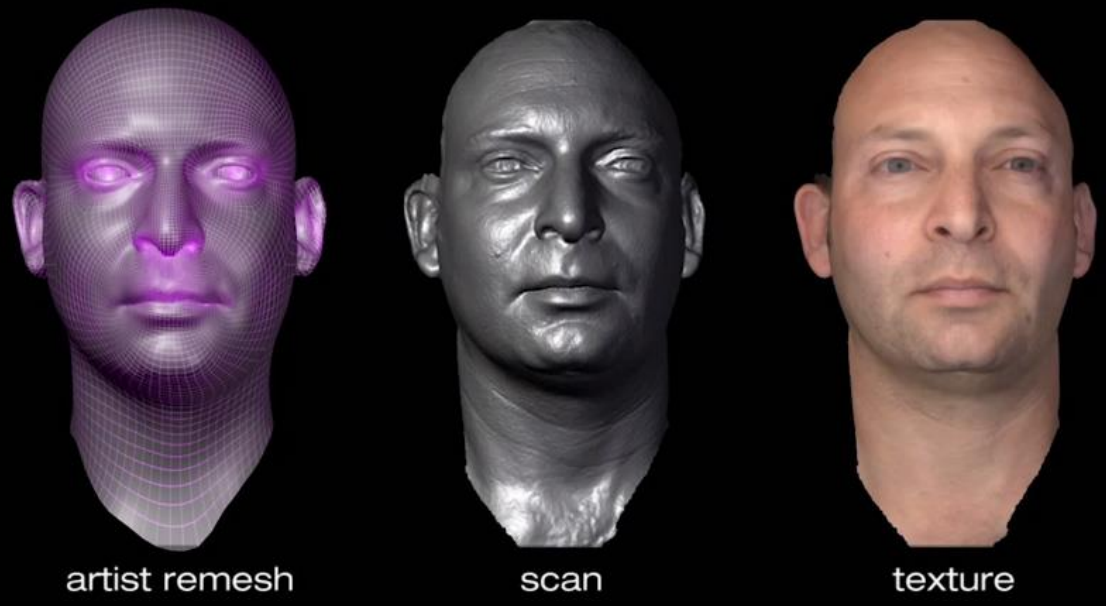


Figure 1: *Our system captures high-fidelity facial performances with realistic dynamic wrinkles and fine-scale facial details.*

Digital Ira

- Lightstage
- Optical flow with diffuse map
- http://www.youtube.com/watch?v=IstcFOGwvU4
- Very even lighting



Modified approach:

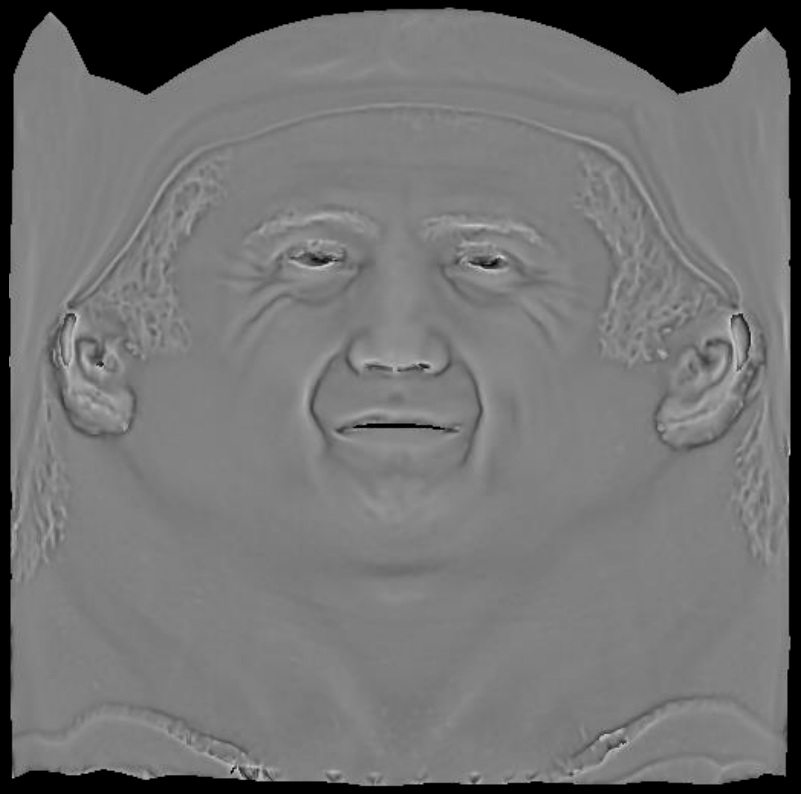
- For each scan, find the nearest scans in terms of curvature.
- From left to right: Smile, Ss, Aa

Modified approach:

- In theory, all scans should converge to neutral
- In reality, not all will converge
  - Will at least converge to “close” scans

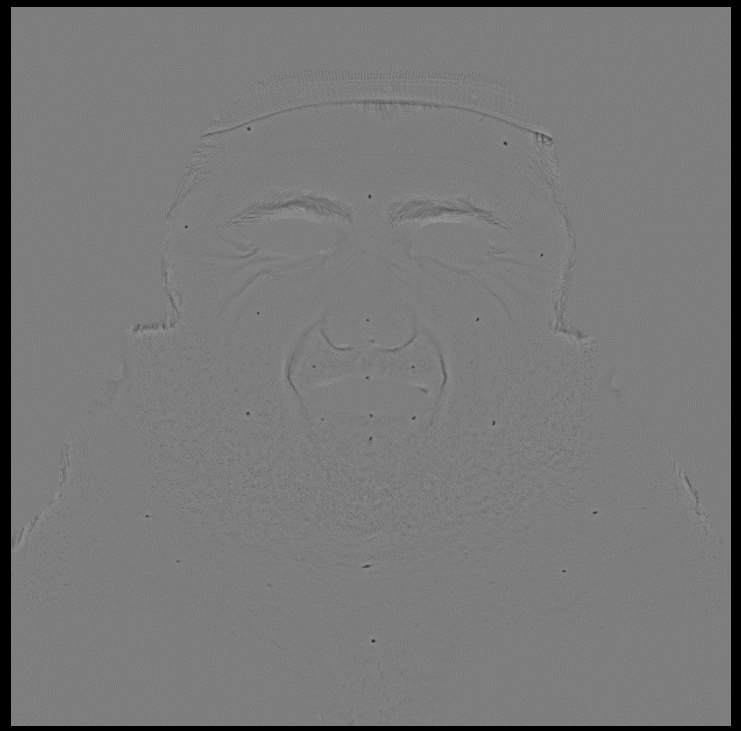
Curvature:

- Export curvature for each scan.
- Also apply some image-based noise reduction.



Diffuse:

- Solving diffuse does not work too well.
- Scanning room is evenly lit, but you will never get it perfect.
- Some directionality in the lighting is built into the scans.
  - Causes optical flow to get confused.
  - Solution: Extract high frequencies from diffuse.



Optical Flow

- On more trick:
  - Solve for the neutral pose as well as the nearest 5
  - Median will ignore it if it does not solve well.
- For three iterations:
  - For each pose:
    - Solve optical flow for curvature for all 5 nearest poses + neutral.
    - Take the median.
  - For each pose:
    - Solve optical flow for high-frequency of the diffuse for all 5 nearest poses + neutral.
    - Take the median

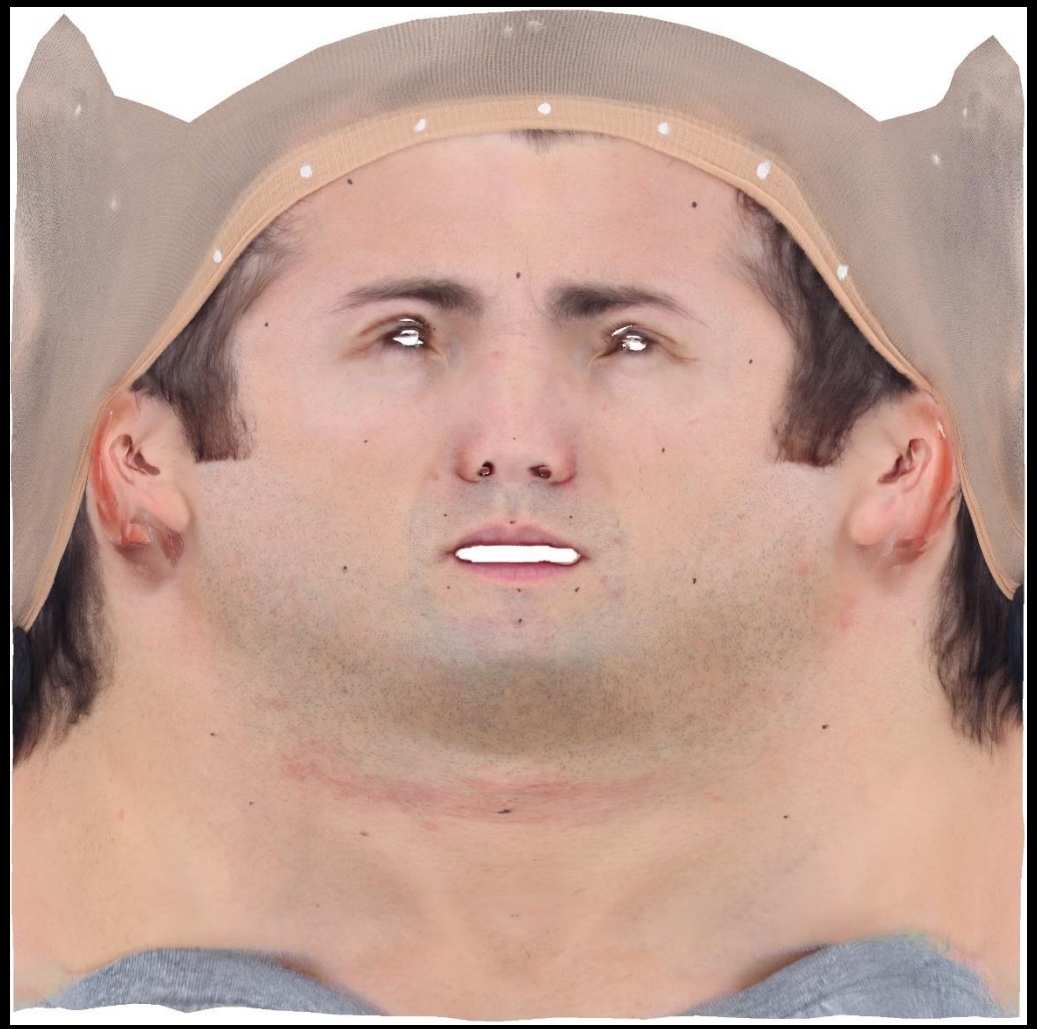
Final Step:

Apply optical flow to each mesh.

Ignore optical flow around eyes, mouth, and back of the head.

Results:

- Before optical flow on left, after on right



Results:

- Before optical flow on left, after on right

Results:

- Before optical flow on left, after on right

Results:

- Before optical flow on left, after on right

Results:

- Before optical flow on left, after on right

Results:

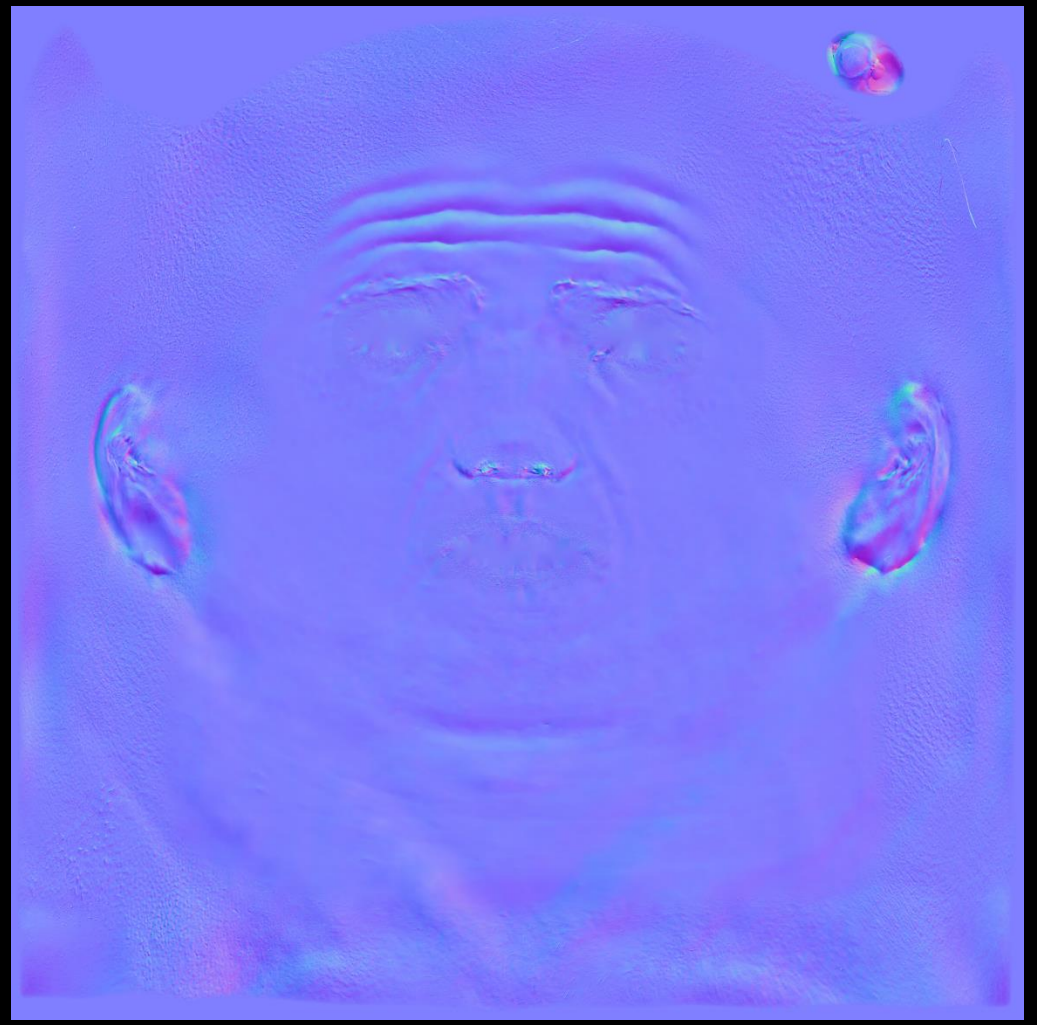
- Before optical flow on left, after on right

Optical Flow:

- Effect is subtle.
- Makes a HUGE difference during compression.

Normals:

- Still need a normal map.
- Custom transfer function
- Run image-space noise reduction

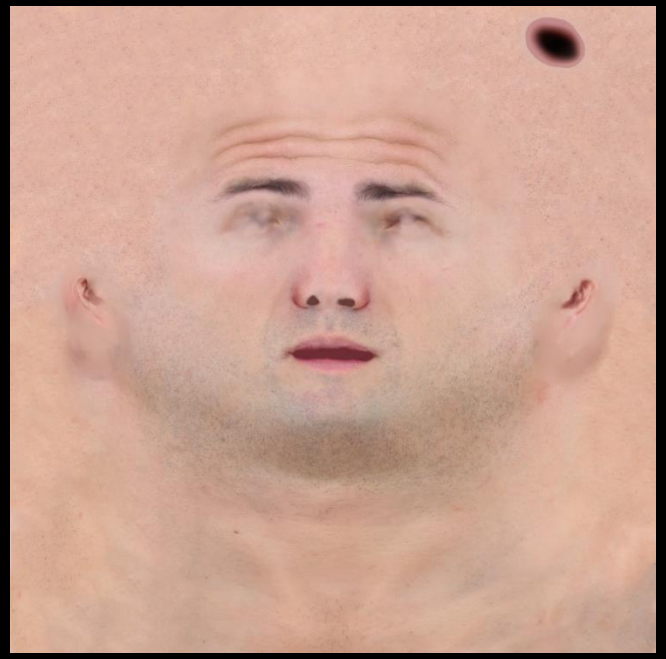


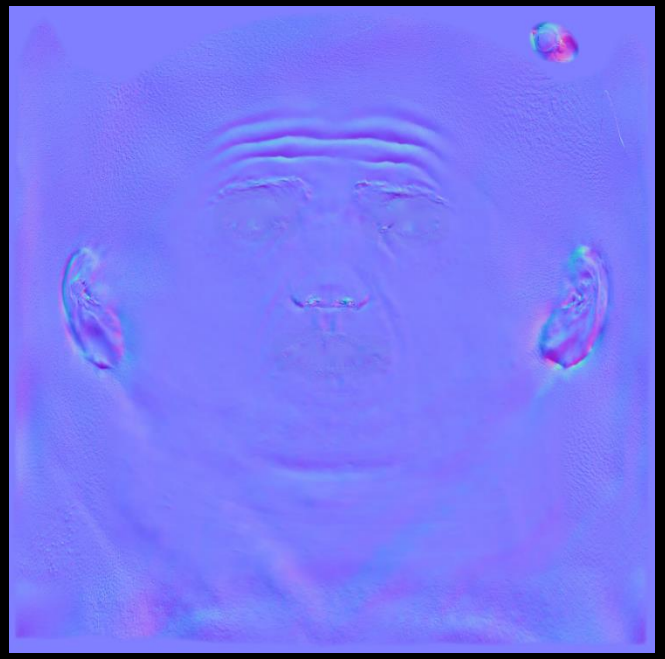
AO:

- We also need AO.
- Custom AO bake
- Use as specular mask

Blendshapes:

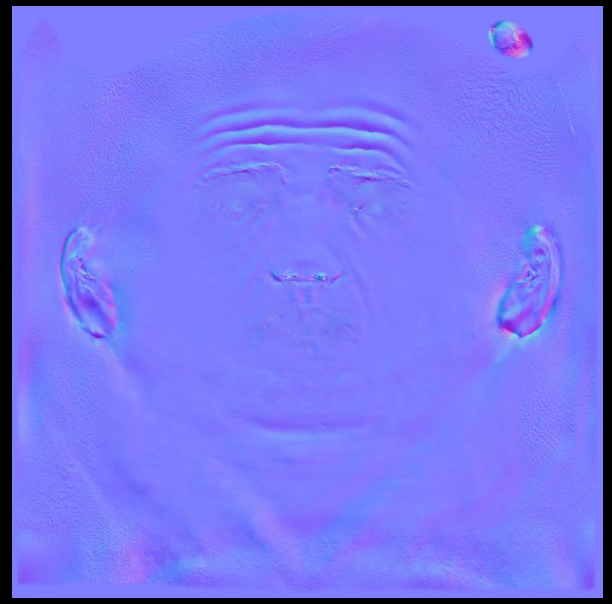
- And that is how to create blendshapes!
- Full pipeline takes about a day or two to run.
  - Could be put on a farm. I just have one machine.
- All maps are processed in 2k, to extract 1k maps from center.
- Have done some 8k tests, maybe later





Blendshapes:

- CPU Time = Cheap
- Artist Time = Expensive
- Very little manual work per-shape
  - About a day or two to align markers
  - Less than a day to manually tweak the lips
  - The rest is just processing
    - And, of course, writing the code
- Fully non-destructive
  - Not an issue to rerun the whole pipeline
  - Need to rerun when the rig updates



Step 2: Compression/Decompression

Recap:

- We have:
  - 70 blendshapes
  - 70 diffuse maps
  - 70 normal maps
  - 70 ao maps
- The shapes are less of an issue.
- The textures are a problem

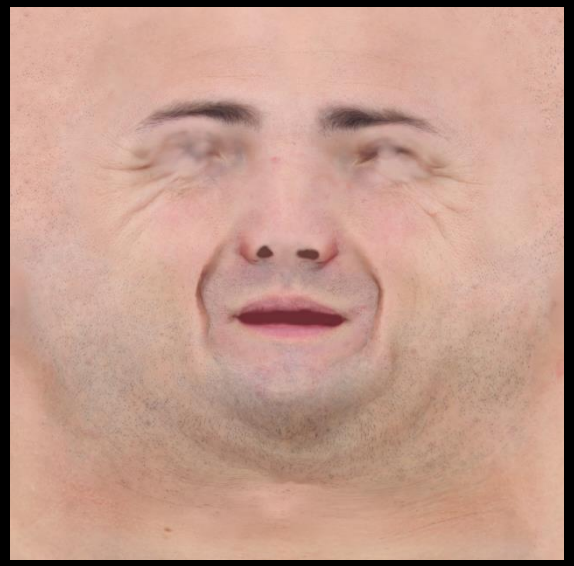
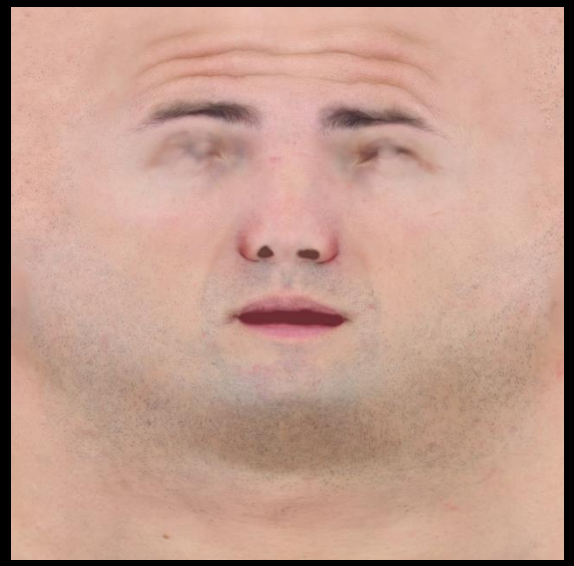
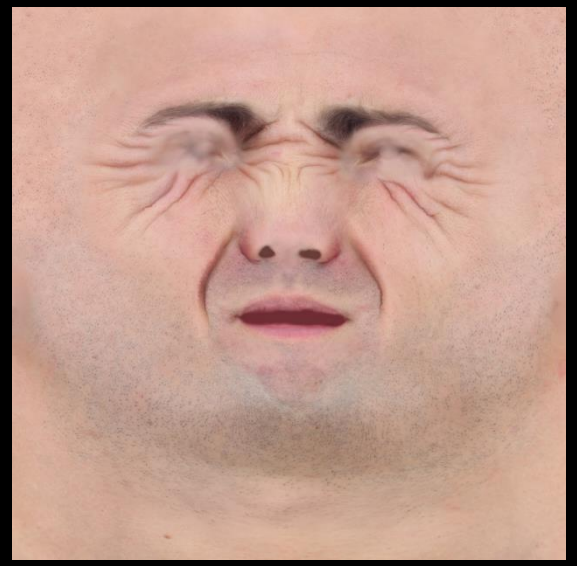
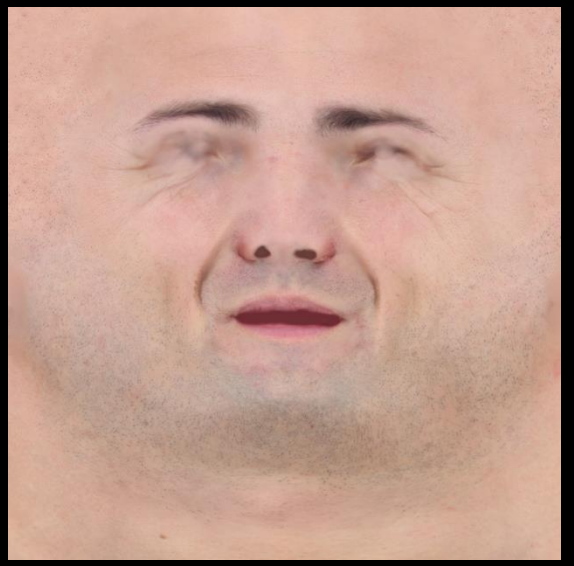
Where have I seen this problem before…

Playable Ucap

- GPU Gems 3: Playable Universal Capture
- Compression was Diffuse only
  - No animated Normal or AO map
- Entire texture was compressed
  - Includes Teeth/Tongue/Mouth Bag
  - Eyes also included

Guide to PCA

- Many Diffuse Textures (70)
  - Tiger Woods had thousands

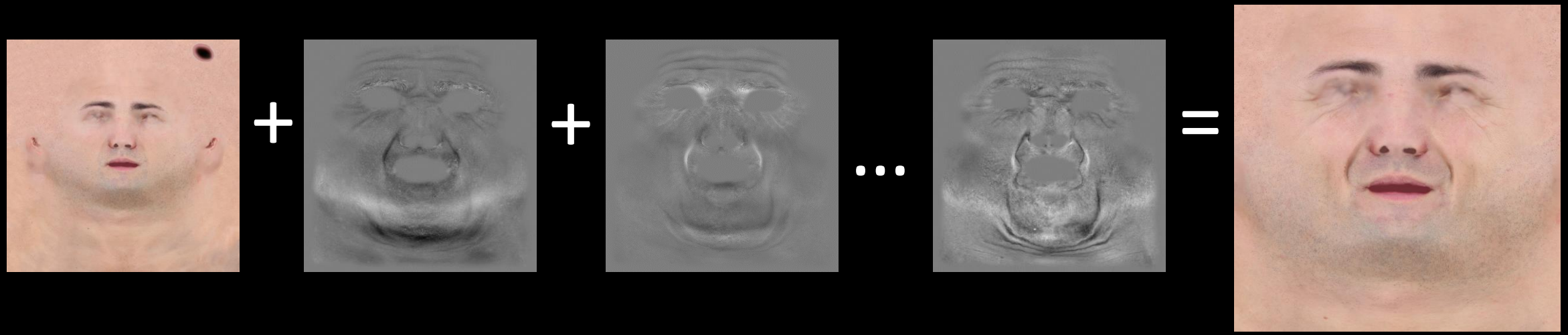





Guide to PCA

- Actually the offset
- Showing green channel
- From left to right:
  - Smile, Neutral, Smile Offset (Smile - Neutral)

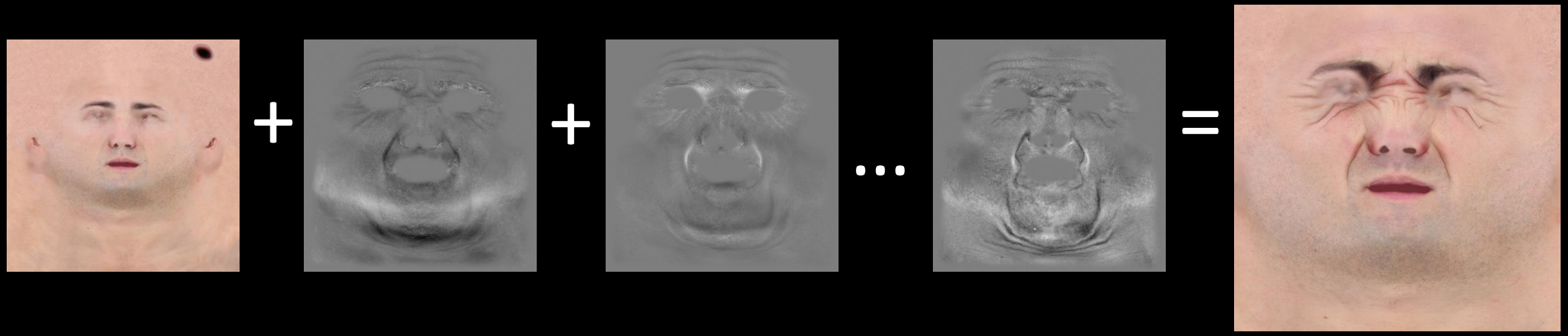
Guide to PCA

- Reconstruction
- Final = w0*img0 + w1*img1 + w2*img2 + … + w11*img11
- Animate by just changing weights (shader constants)



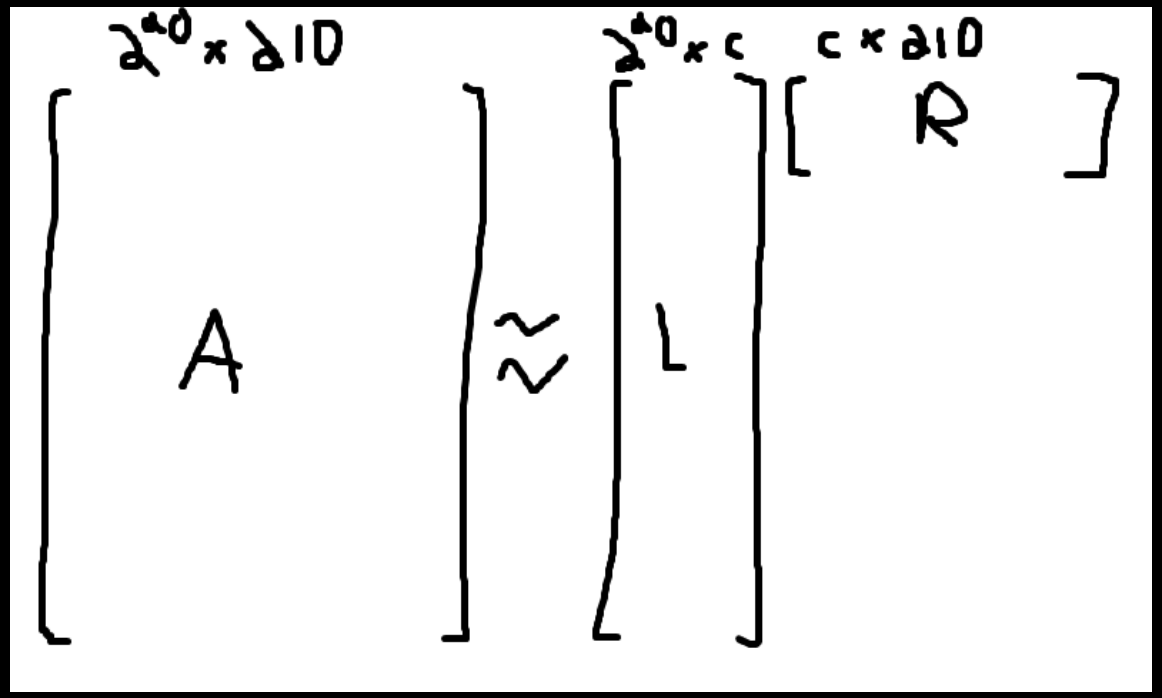
Guide to PCA

- Reconstruction
- Final = w0*img0 + w1*img1 + w2*img2 + … + w11*img11
- Animate by just changing weights (shader constants)



Calculating PCA

- Could use SVD
- SVD will solve all 210 columns
- Then we chop off all but the first 12



Problems with SVD

- Did this on Tiger Woods
- Dedicated machine with lots of RAM
- Really slow. Overnight.
- Debugging nightmare.
- What if it breaks? Hope it doesn’t happen.

Iterative Calculation

- Not too difficult
- Simple algorithm
- Stability not an issue
  - At least not with 12 components

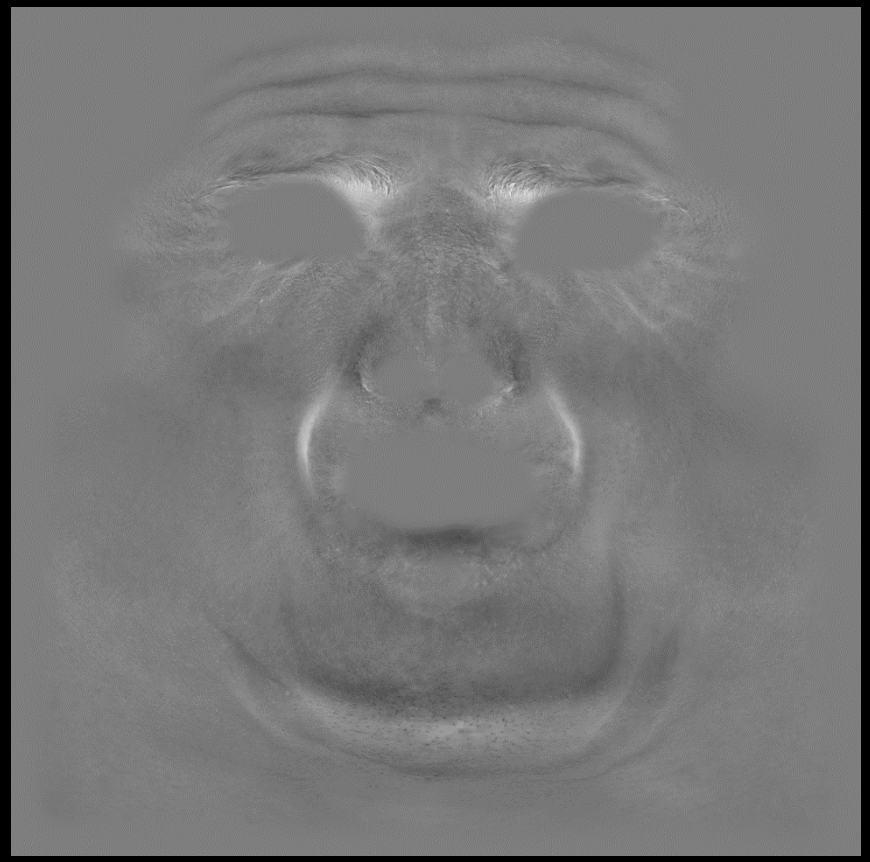
PCA

- Algorithm: From Wikipedia
- http://en.wikipedia.org/wiki/Principal_component_analysis
- Calculate first eigenvector
- After each eigenvector, subtract out the projection of each vector into the eigenvector

$\mathbf{r}$ = a random vector of length p
do c times:
    $\mathbf{s} = 0$ (a vector of length p)
    for each row $\mathbf{x} \in \mathbf{X}$
        $\mathbf{s} = \mathbf{s} + (\mathbf{x} \cdot \mathbf{r})\mathbf{x}$
    $\mathbf{r} = \frac{\mathbf{s}}{|\mathbf{s}|}$
return $\mathbf{r}$

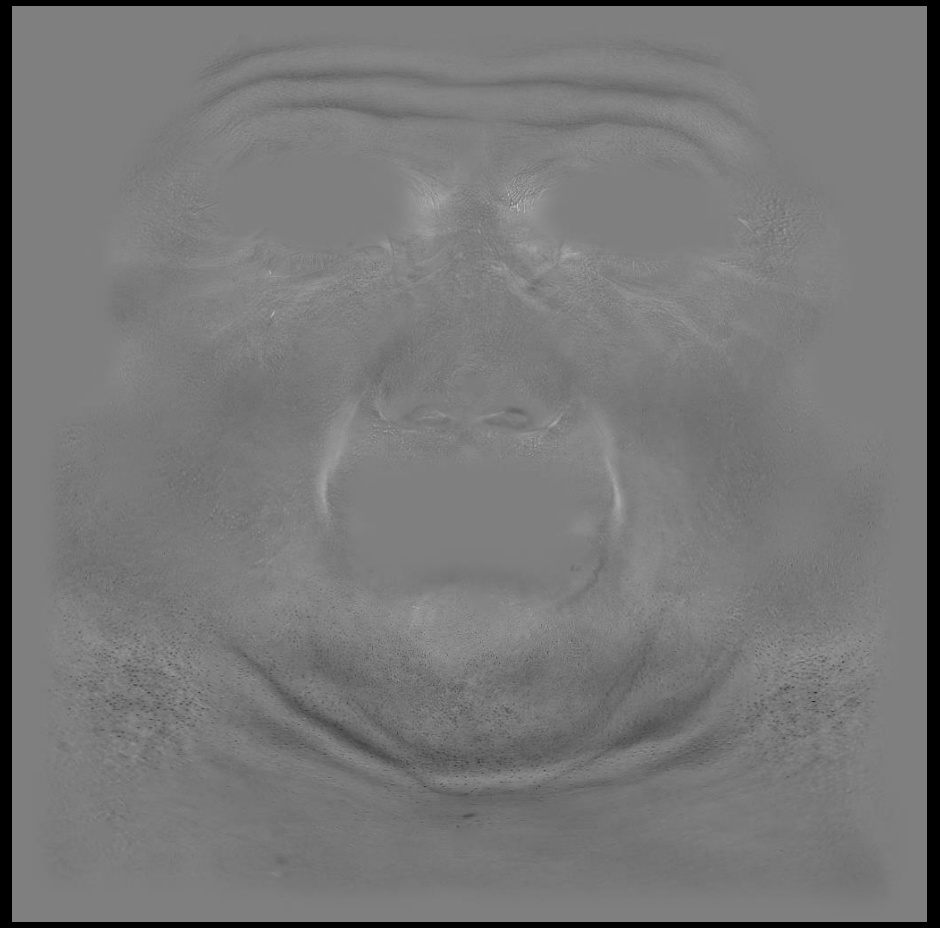
Regions

- Calculate each eigenvector normally



Compression Problems

- Linear whole image
- Hard to isolate regions



Compression Problems

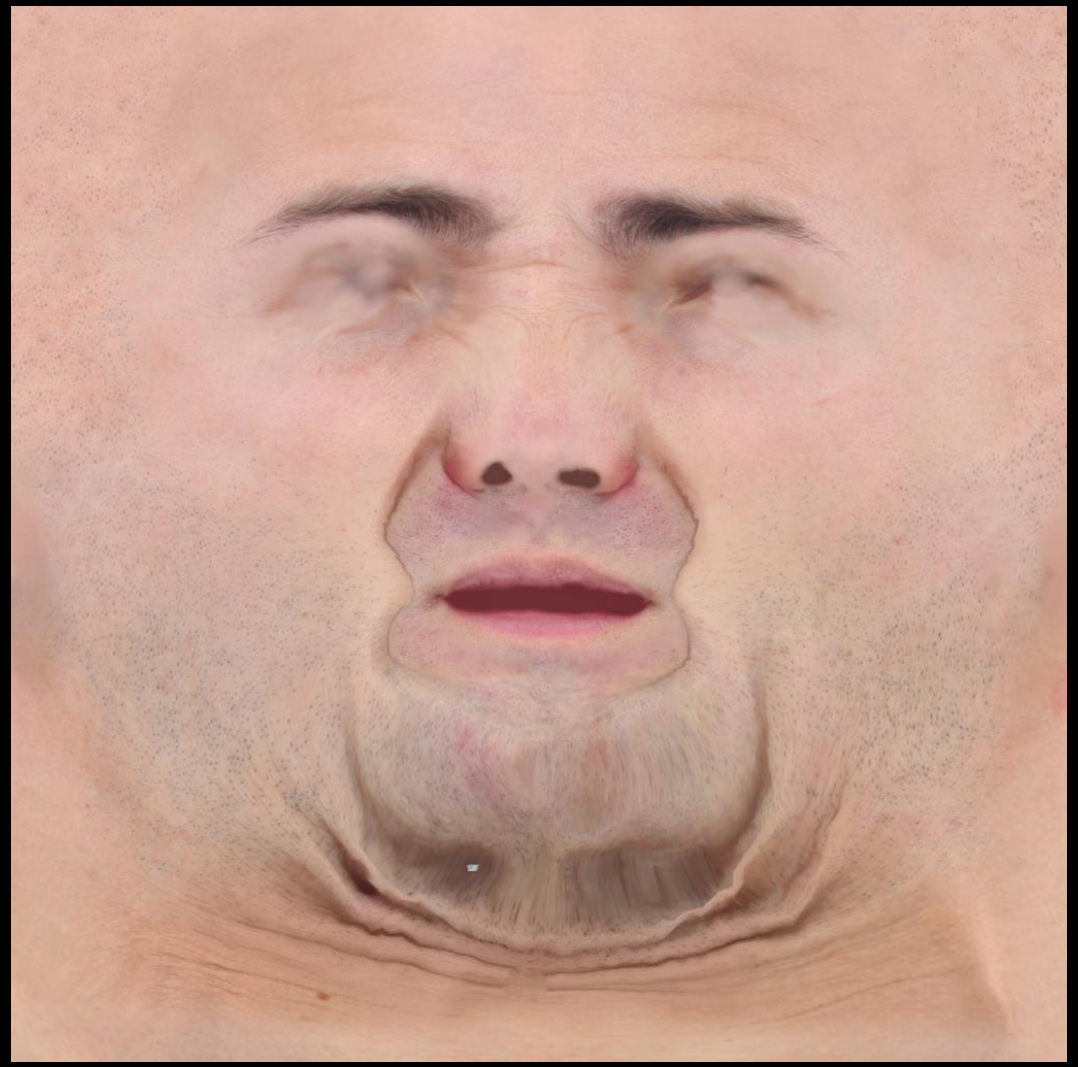
- Stretch
- Wants both

Compression Problems

- FACS 02
- Want just the forehead, not under the chin.

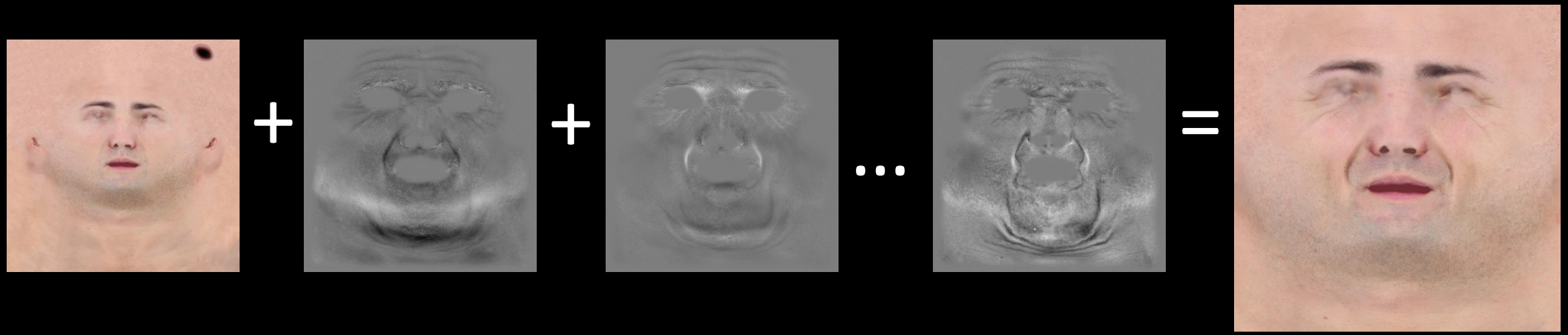
Compression Problems

- FACS 27
- Want just the under the chin, not the forehead.



Corrective shapes

- Not possible to only take part of the region
- Could we fix this?

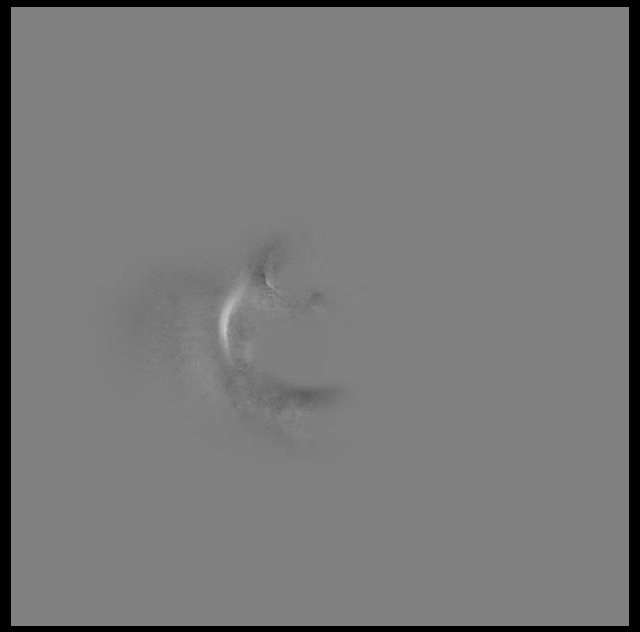


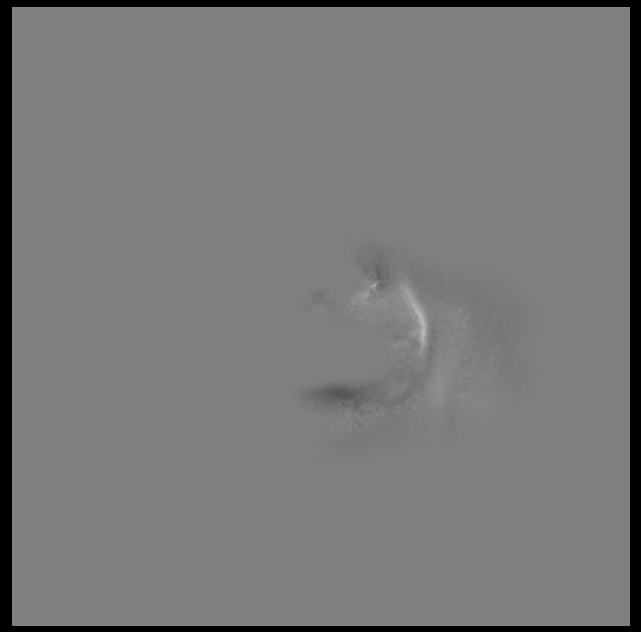
PCA

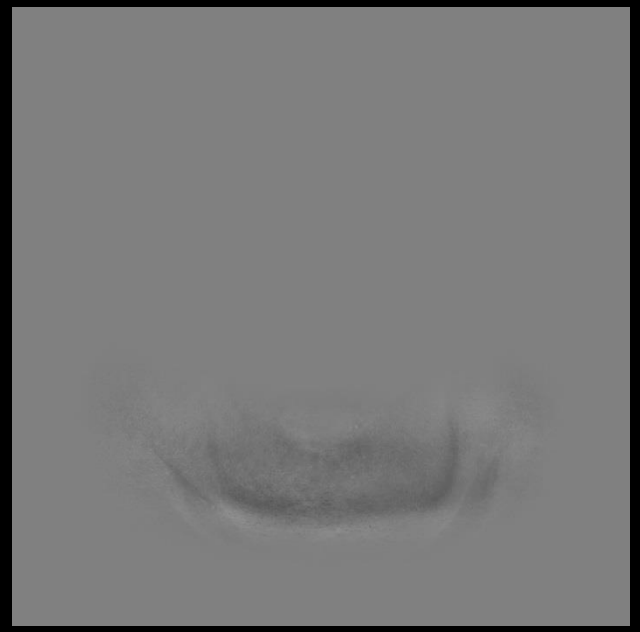
- Instead of choosing weights for each texture, you choose weights for each region
- Adds a little bit to decompression cost
- Greatly helps the compression

Regions

- Split regions into PCA components
- Mouth left, mouth right, chin/neck

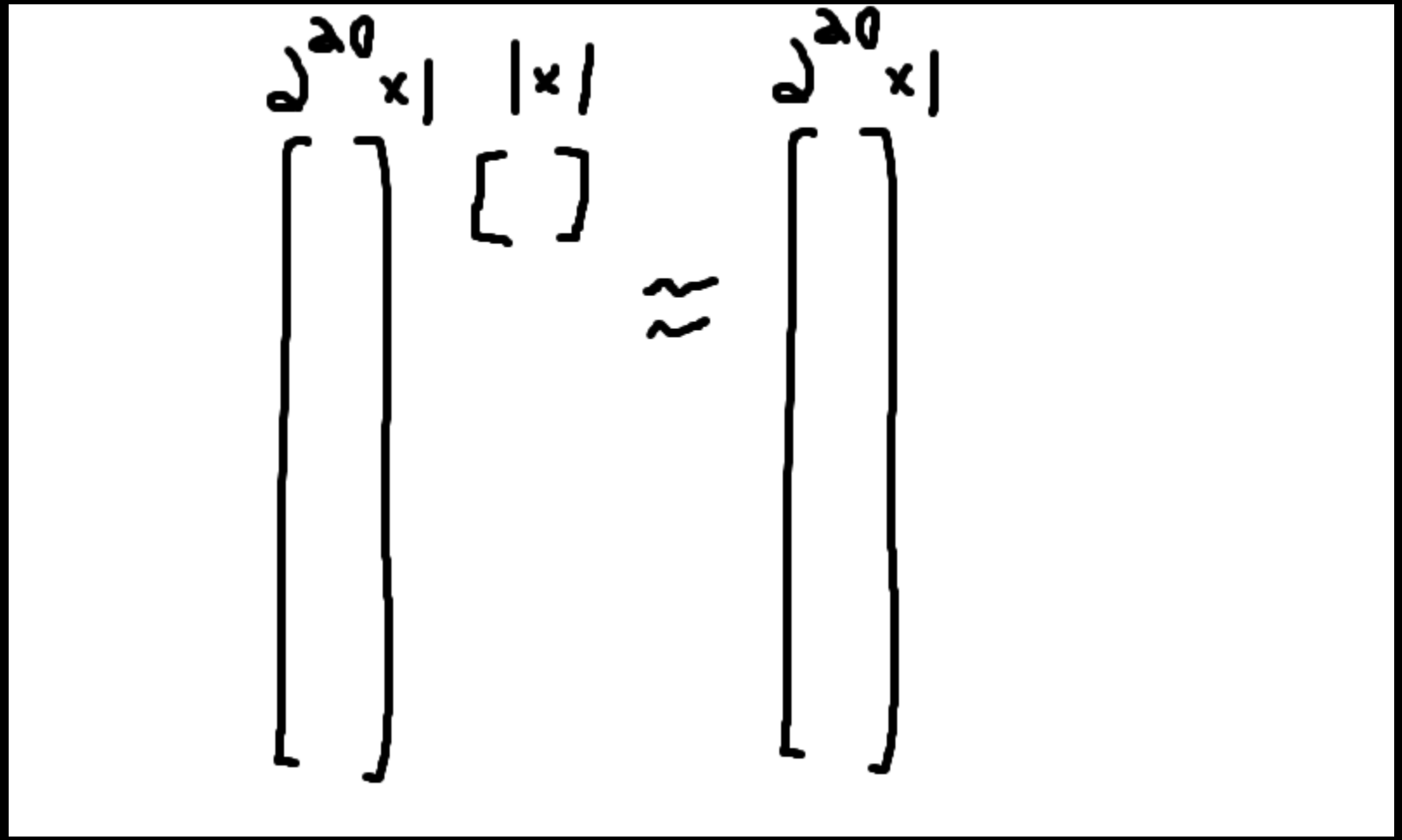






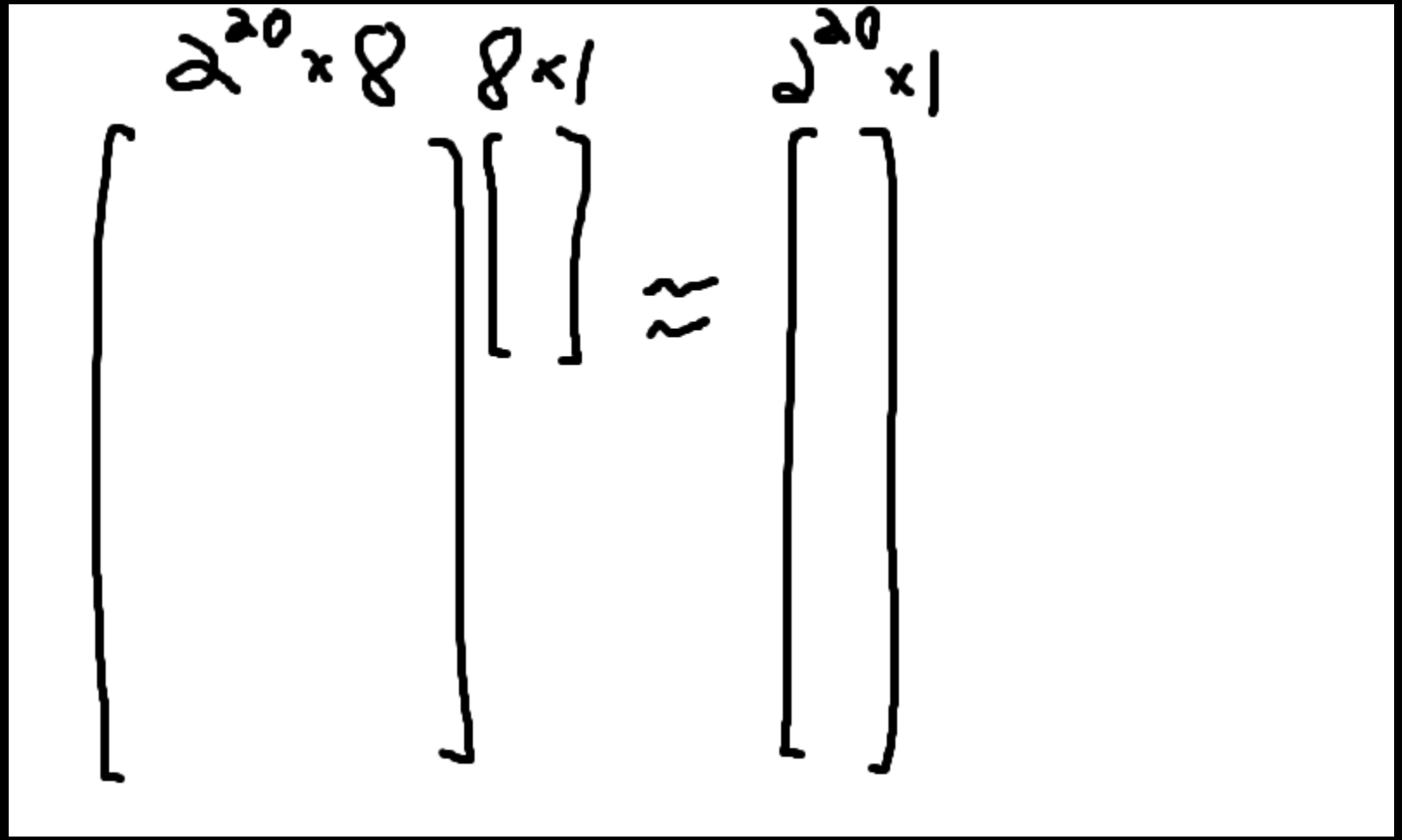
Regions

- Typical PCA, find scalar (1x1) that gets you as close as possible.
- Remove, leaving residue



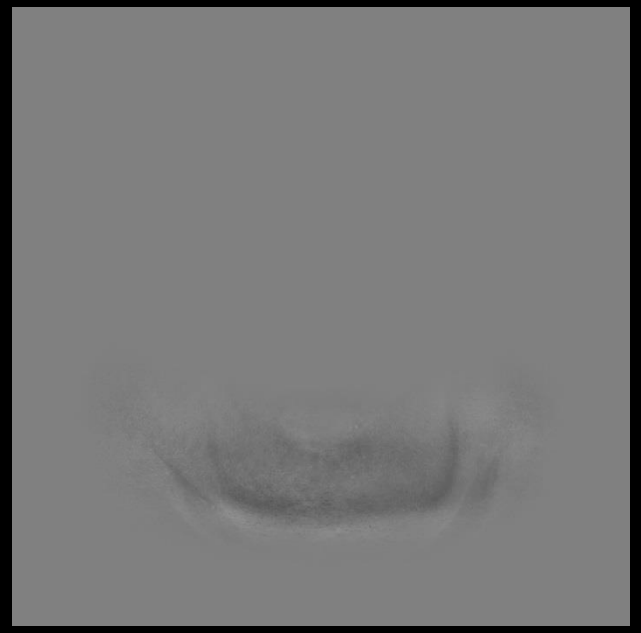
Regions

- Find optimal weights for 8 separate masked textures
- Least squares



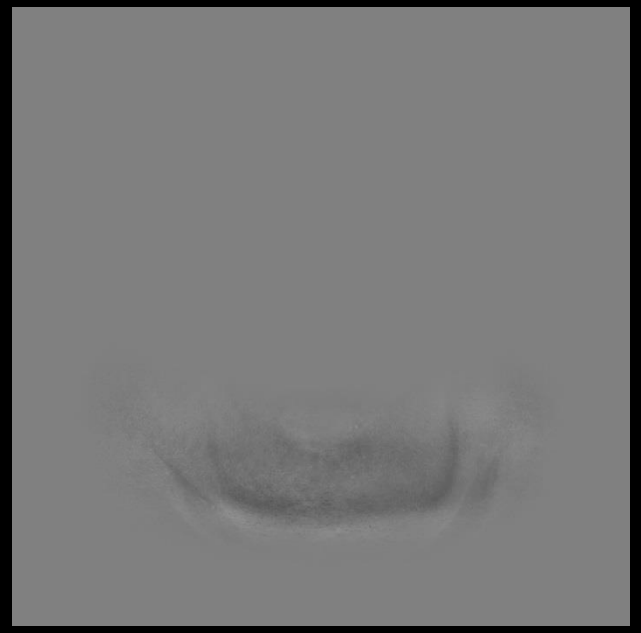
Compression Problems

- Stretch
- Wants it



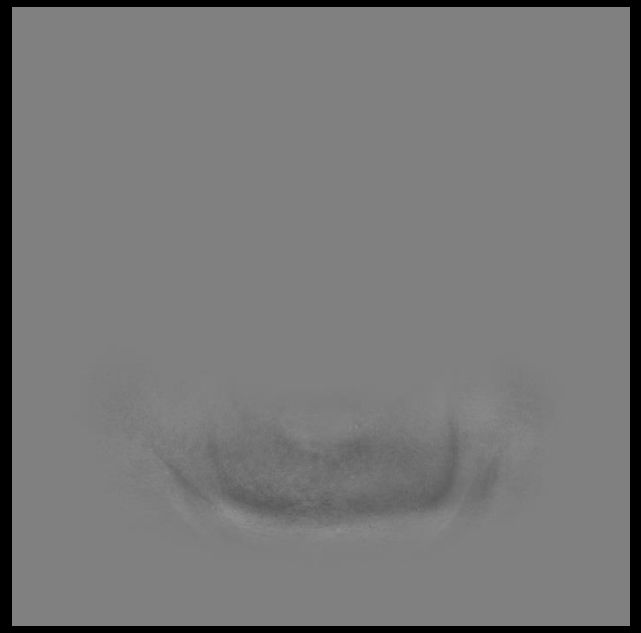
Compression Problems

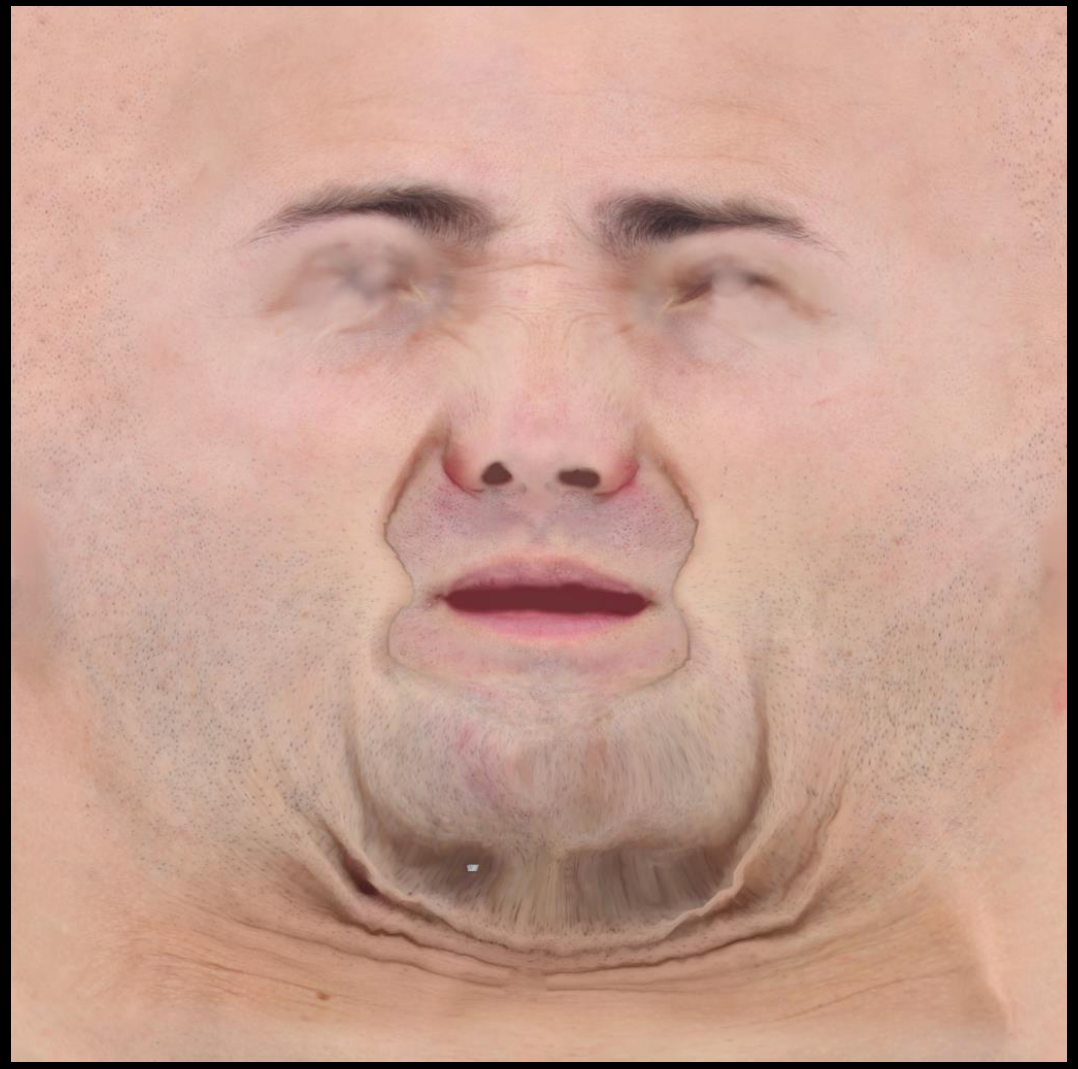
- FACS 02
- Want just the forehead, not under the chin.



Compression Problems

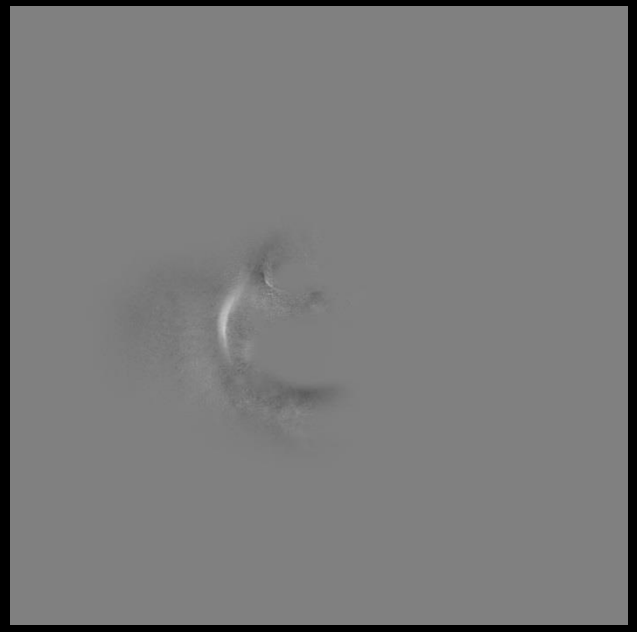
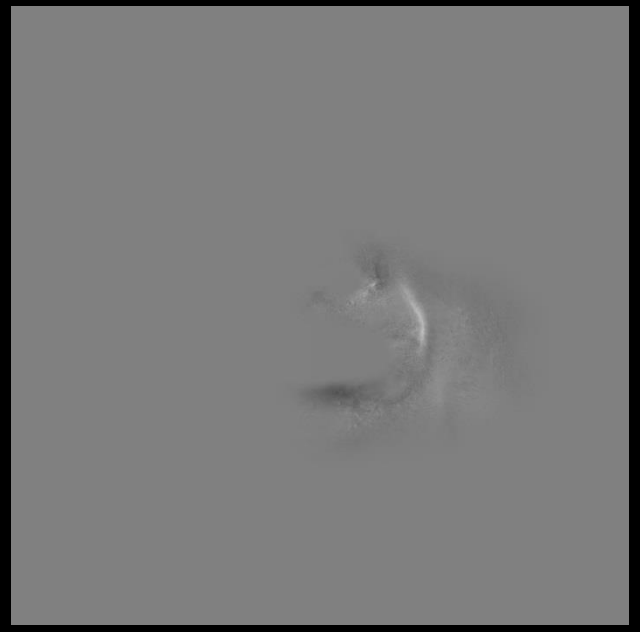
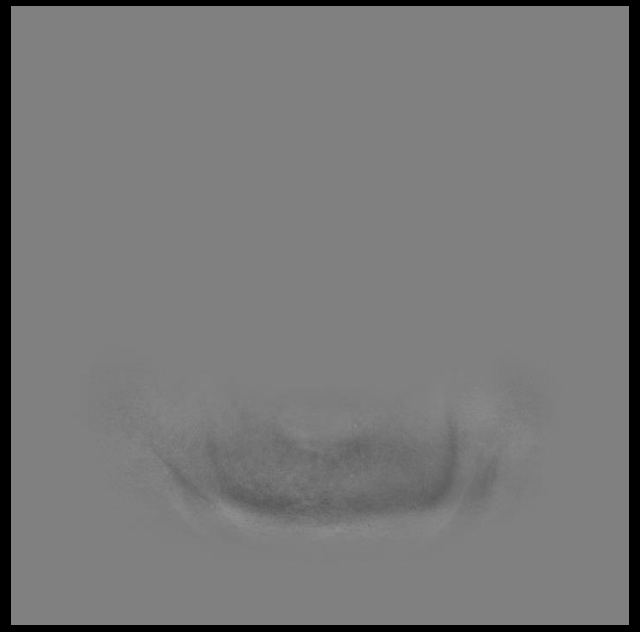
- FACS 27
- Want just the under the chin, not the forehead.





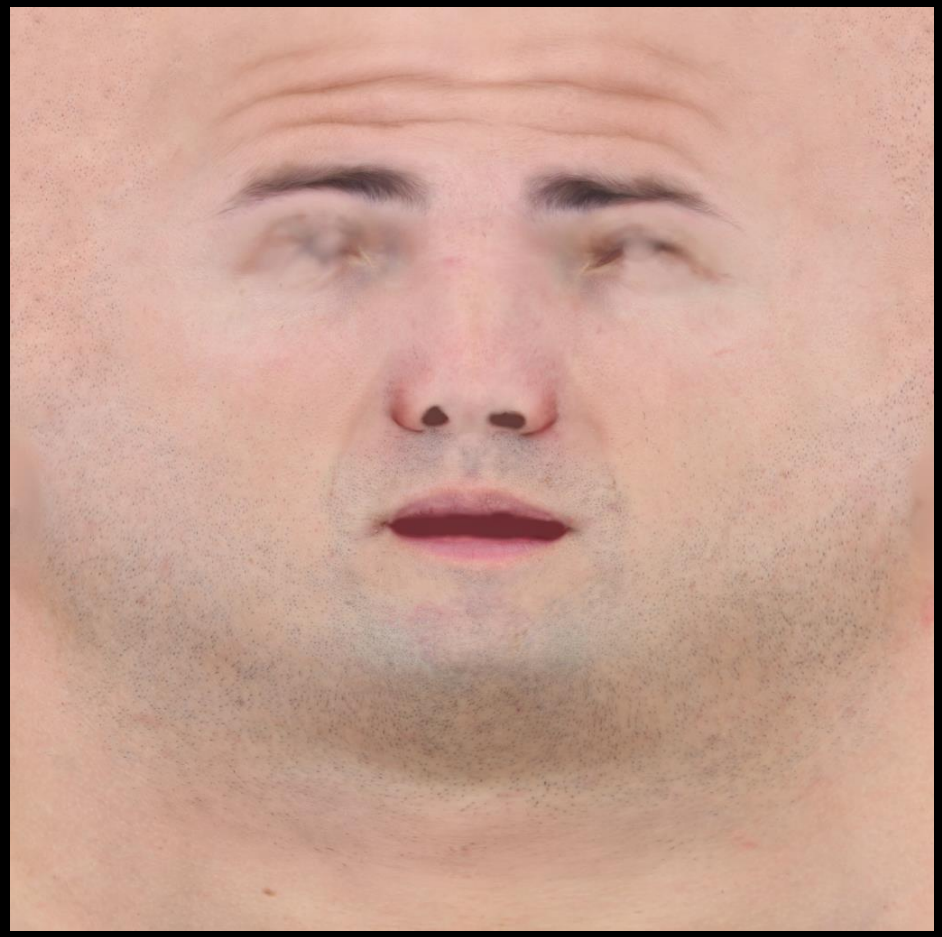
Regions

- By isolating which region get which PCA components, solve is much better
- Especially for first few components

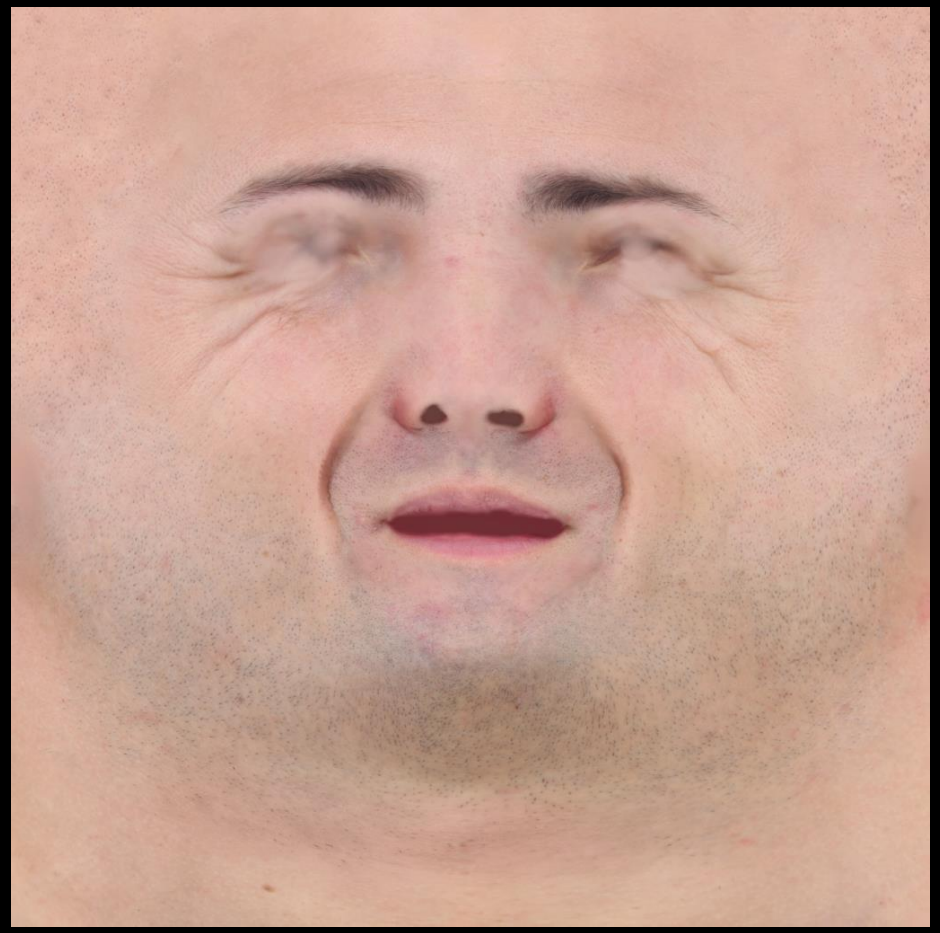
Results with 12 components

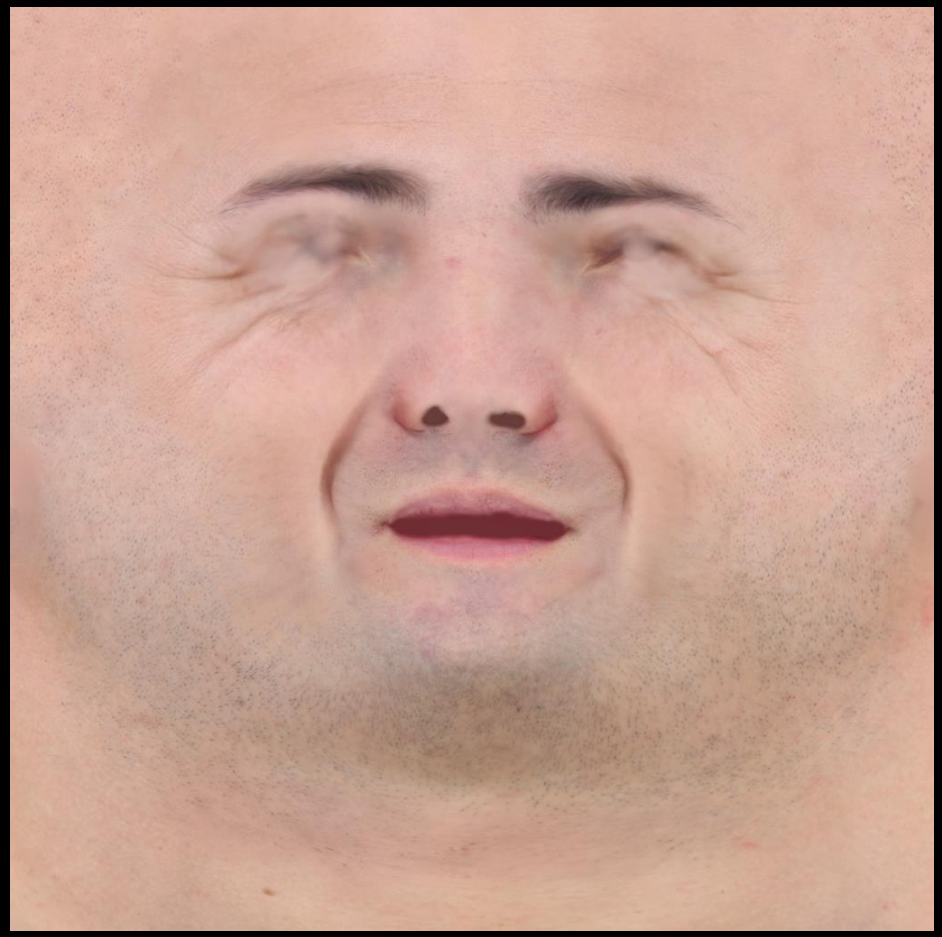
Left is before, right is after.



Results with 12 components

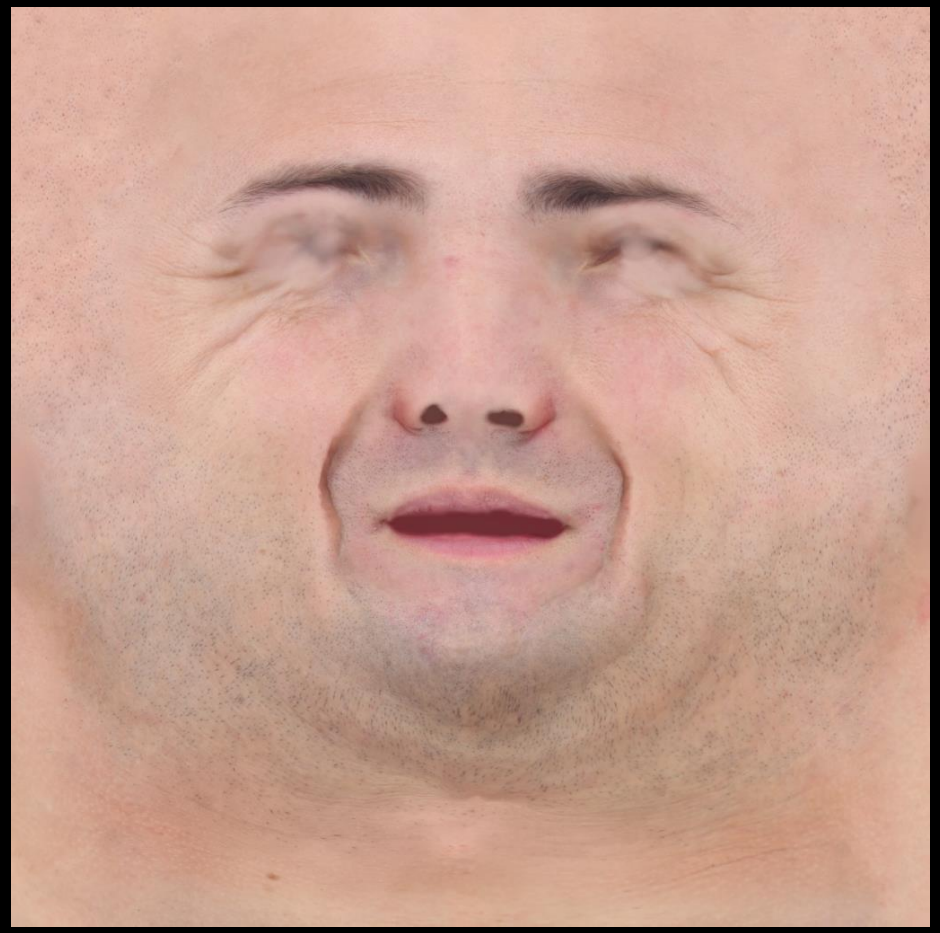
Left is before, right is after.

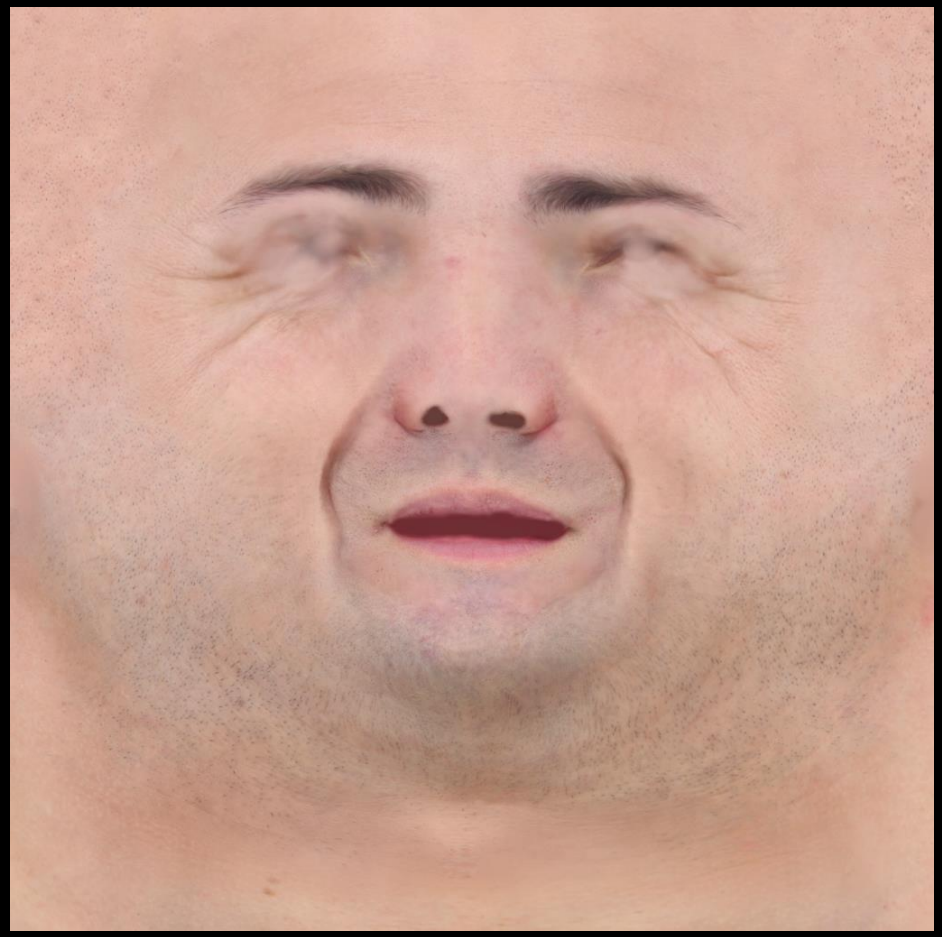




Results with 12 components

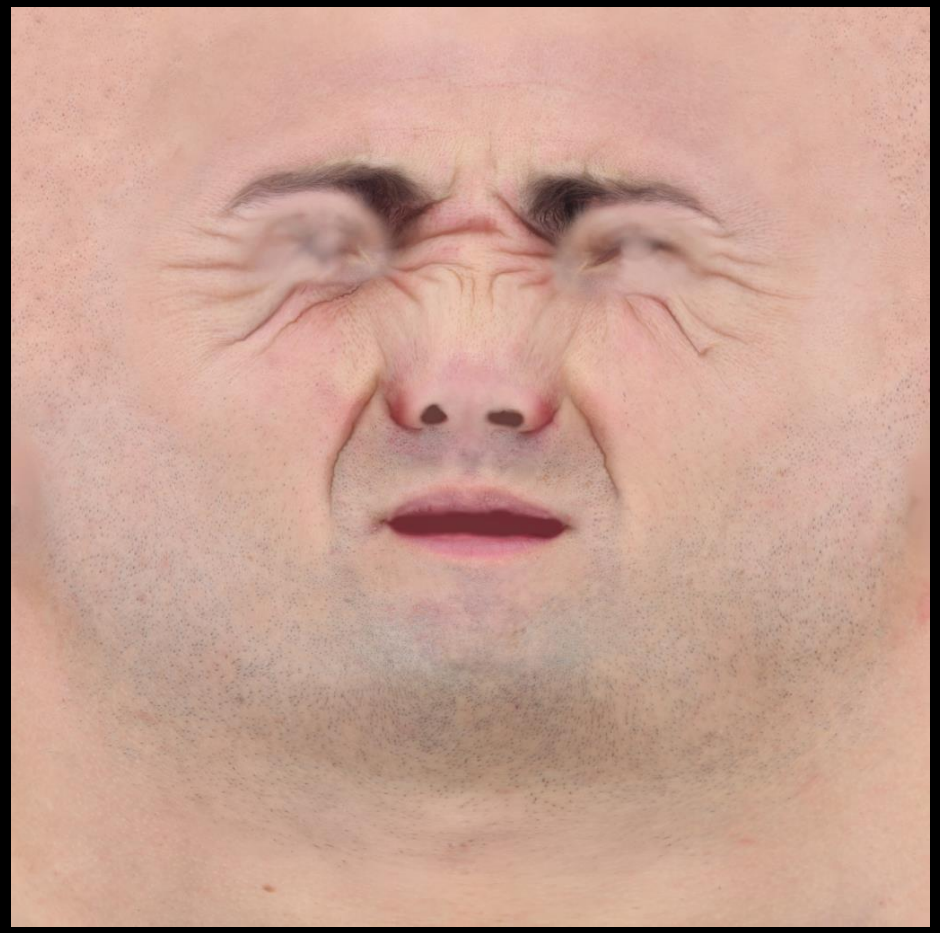
Left is before, right is after.

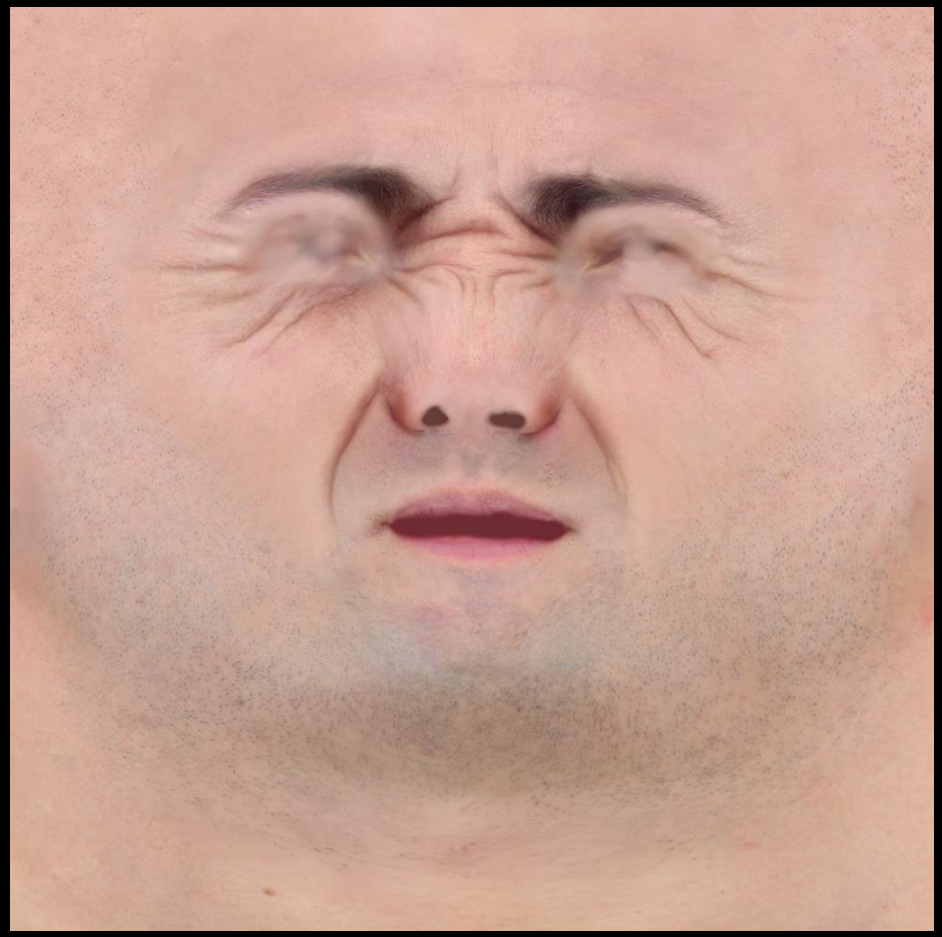




Results with 12 components

Left is before, right is after.



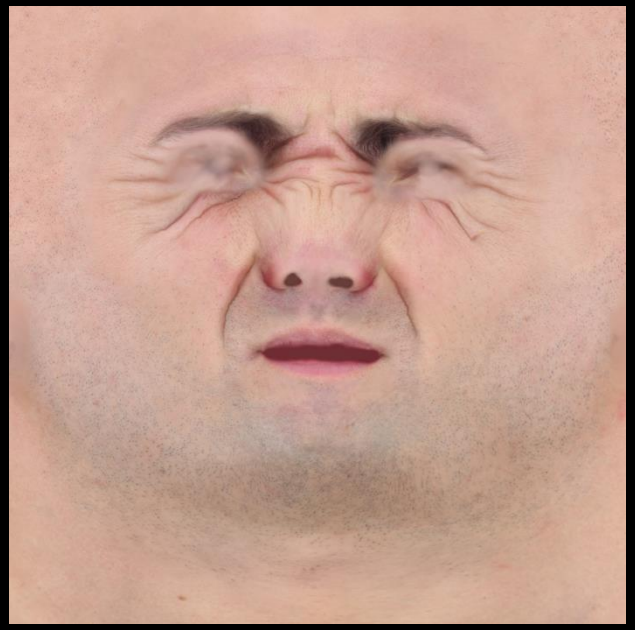


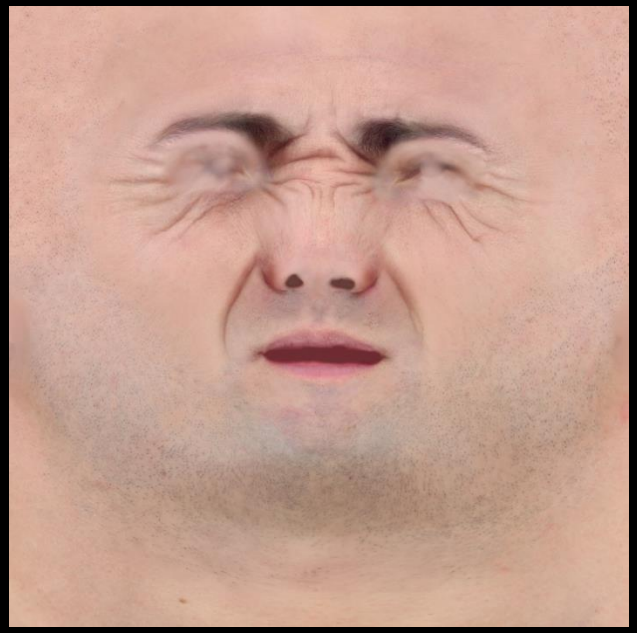
PCA

Definitely loses some detail.

Could be much, much worse.

When there is texture swimming, it loses detail fast.



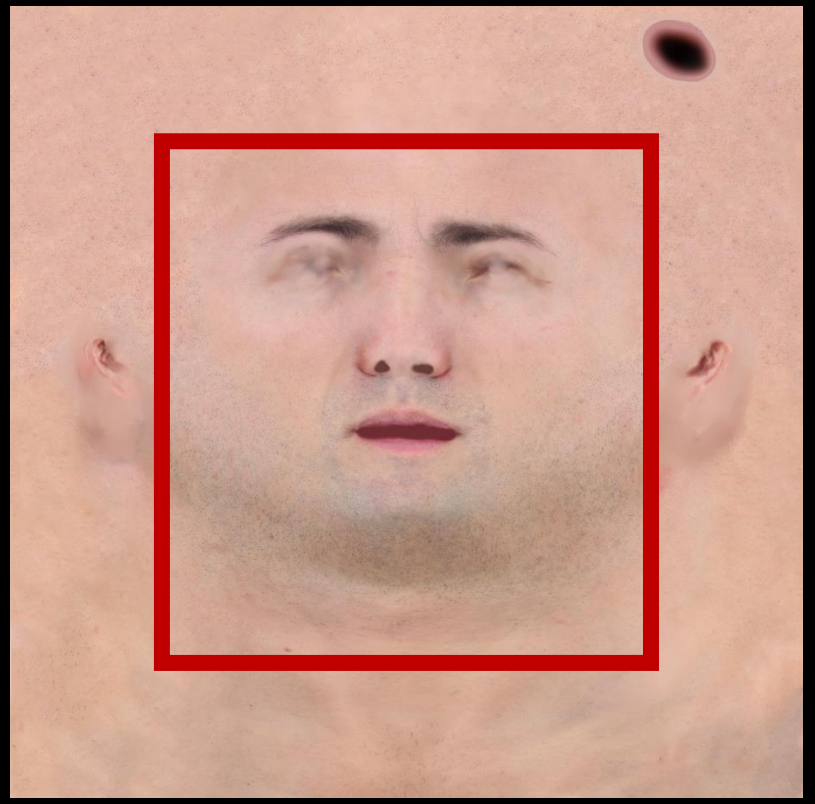


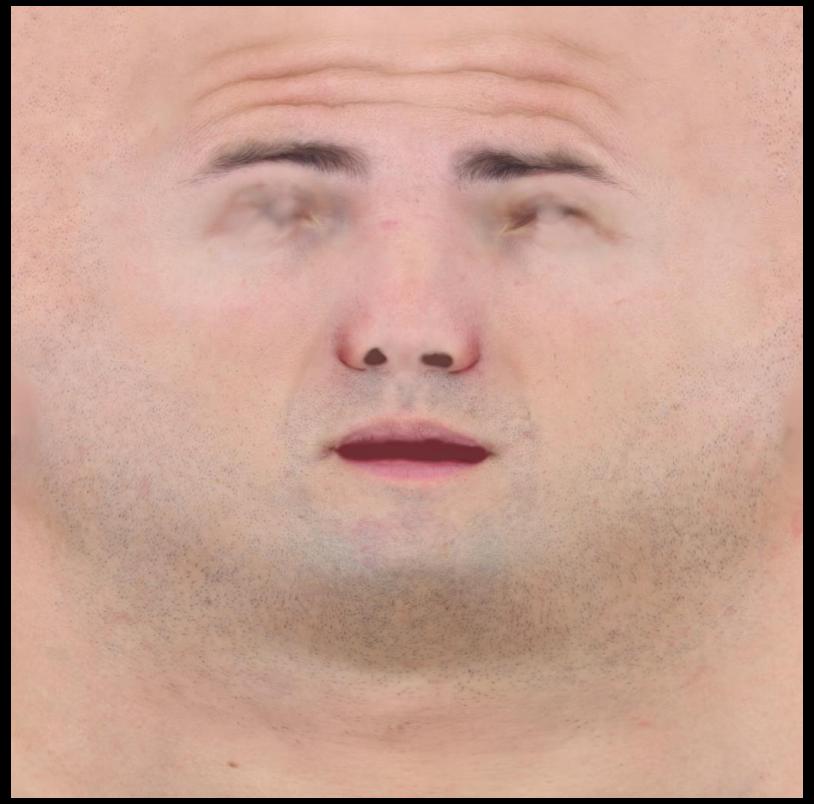
Shader:

```
float4 region0 = g_txPcaRegion0.Sample( g_samLinearClamp, input.TextureUV ).xyzw;
float4 region1 = g_txPcaRegion1.Sample( g_samLinearClamp, input.TextureUV ).xyzw;

float regionData[8] = { region0.x, region0.y, region0.z, region0.w, region1.x, region1.y, region1.z, region1.w };

[unroll] for (iter = 0; iter < 8; iter++)
{
    float weight = regionData[iter];
    diffR0 += weight * g_pcaMaskDiffR0[iter];
    diffR1 += weight * g_pcaMaskDiffR1[iter];
    diffR2 += weight * g_pcaMaskDiffR2[iter];

    diffG0 += weight * g_pcaMaskDiffG0[iter];
    diffG1 += weight * g_pcaMaskDiffG1[iter];
    diffG2 += weight * g_pcaMaskDiffG2[iter];

    diffB0 += weight * g_pcaMaskDiffB0[iter];
    diffB1 += weight * g_pcaMaskDiffB1[iter];
    diffB2 += weight * g_pcaMaskDiffB2[iter];
}

float3 baseDiff = g_txDiffuse.Sample( g_samLinearClamp, input.TextureUV ).rgb;

float4 comp0 = g_txPcaDiff0.Sample( g_samLinearClamp, pcaUv ).rgba * 2.0f - 1.0f;
float4 comp1 = g_txPcaDiff1.Sample( g_samLinearClamp, pcaUv ).rgba * 2.0f - 1.0f;
float4 comp2 = g_txPcaDiff2.Sample( g_samLinearClamp, pcaUv ).rgba * 2.0f - 1.0f;

ret.x = baseDiff.x + dot(diffR0,comp0) + dot(diffR1,comp1) + dot(diffR2,comp2);
ret.y = baseDiff.y + dot(diffG0,comp0) + dot(diffG1,comp1) + dot(diffG2,comp2);
ret.z = baseDiff.z + dot(diffB0,comp0) + dot(diffB1,comp1) + dot(diffB2,comp2);

ret = pow(saturate(ret),2.2f);
```

Old-School UVs

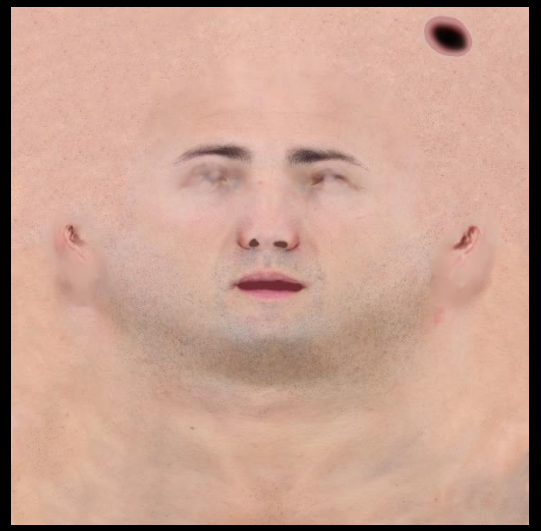
- Flattened UVs, optimized for face.
- Replace the middle with animated textures.
  - Animated texture has good layout and minimal stretching
  - Back of head has major stretching, but we don’t care





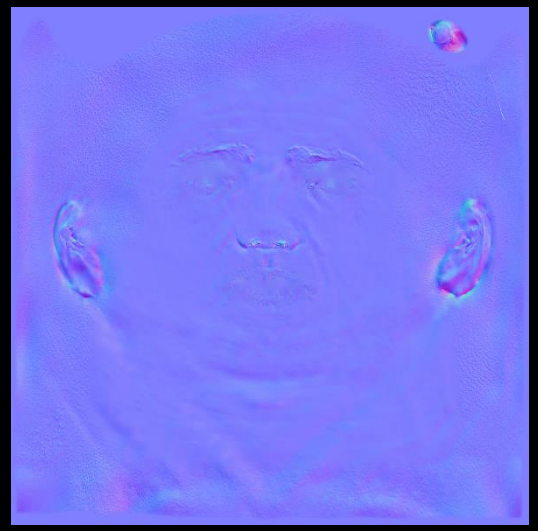
PCA Notes

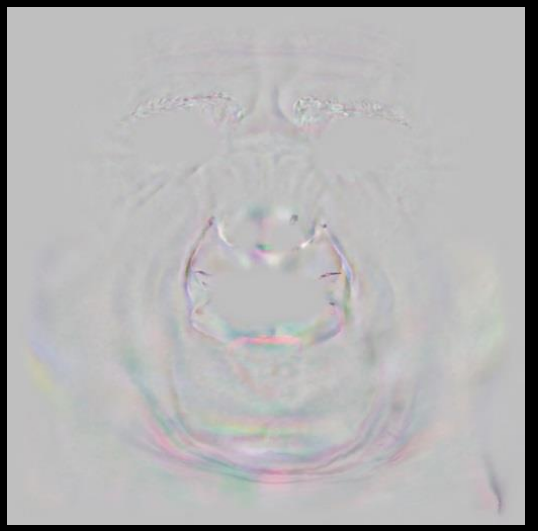
- Use zoomed neutral expression as mean
- 70 diffuse maps as 4, 1k maps
  - 1 regular and 3 zoomed
- Much cheaper than 70, 2k maps!
- Same memory as a single 2k texture




PCA Notes

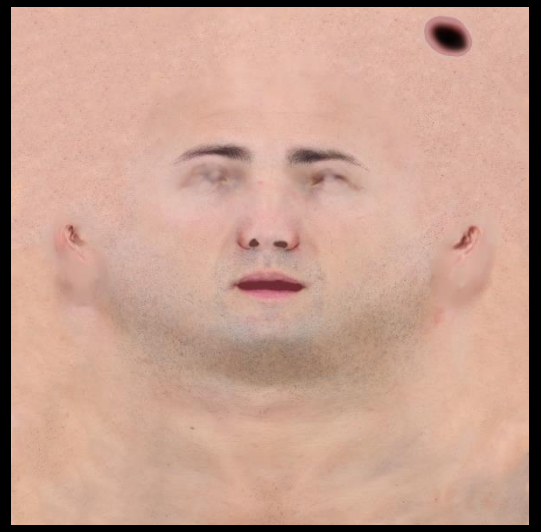
- Normal/AO uses same method, zoom maps are 512
- Actual data is 512
  - You could do larger
- AO base is stored in alpha of Normal, but has separate PCA tex



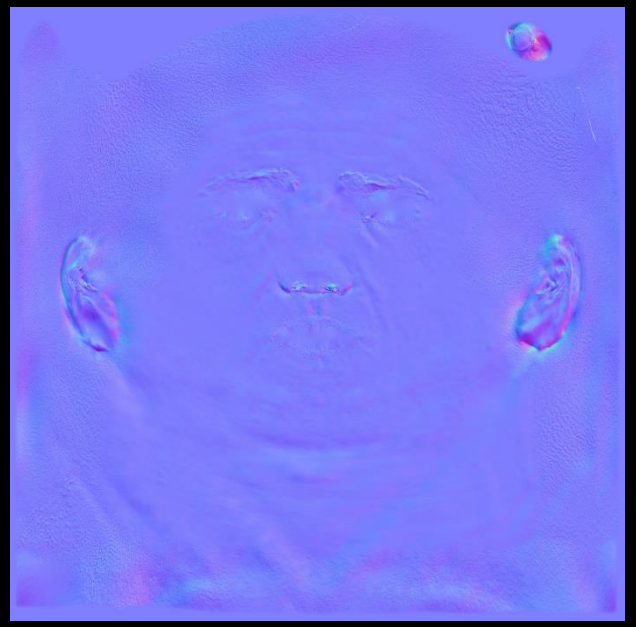
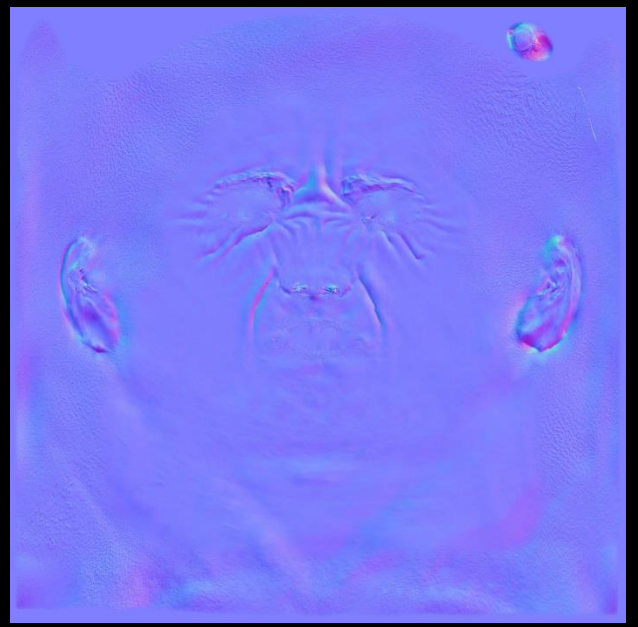
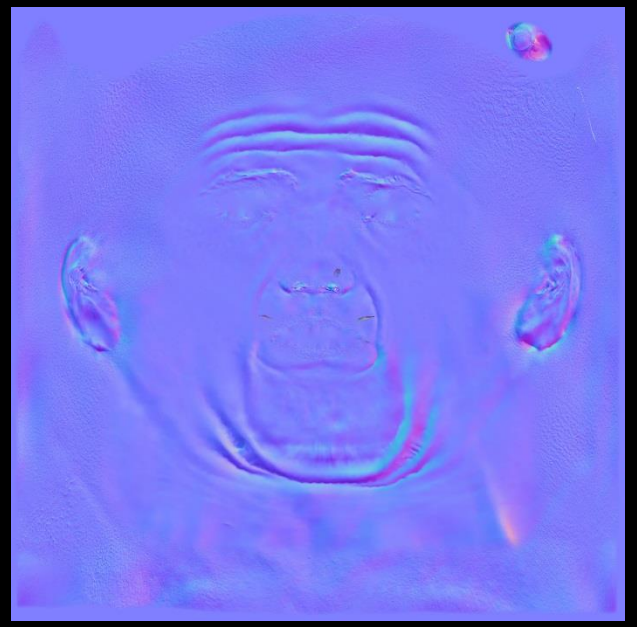
Total Diffuse/Normal/AO Texture Memory

- Diffuse, Normals, AO, + PCA for all 3
  - Plus tiny regions weights
- Everything BC7 compressed
  - Custom encoder
- Total, 8.7 MB
  - Not including teeth/tongue/eyes/stretch discussed later




PCA Notes

- Could be used for standard wrinkle normal maps
- XYZ have good coherency
- Single 2k map is 5.33MB with BC7
- Neutral, Compress, Stretch

Could be useful for other things

- Easy to blend for LOD
  - Scale weights to 0
- Easy to scale quality
  - 12 components is arbitrary
- Easy decompression
  - Free sampling, mipmapping, etc.
- Clint Hanson had water example in 2006
- Baked lightmaps for time of day?
  - Bake out every time of day and compress with PCA?

Hooray, blending!

Hooray, blending!

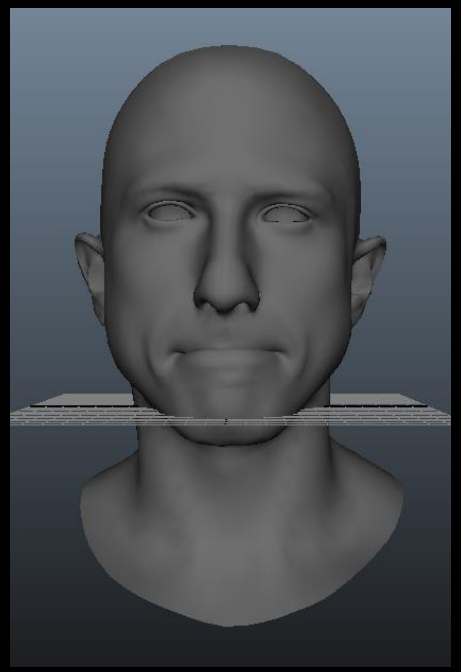
Hooray, blending!

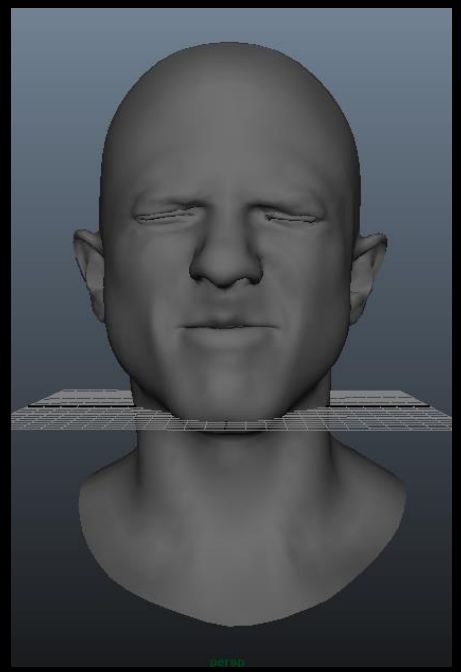
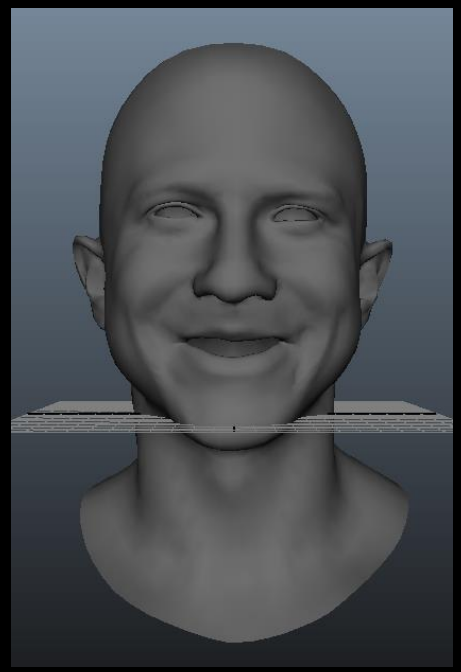
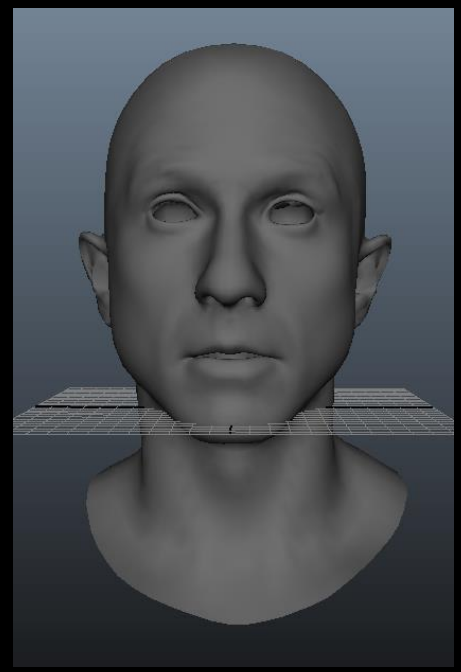
Hooray, blending!

Part 3: Driving with Mocap

So we have blendshapes…

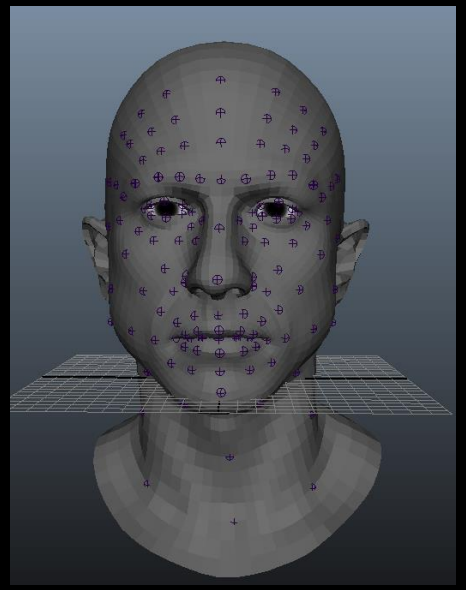
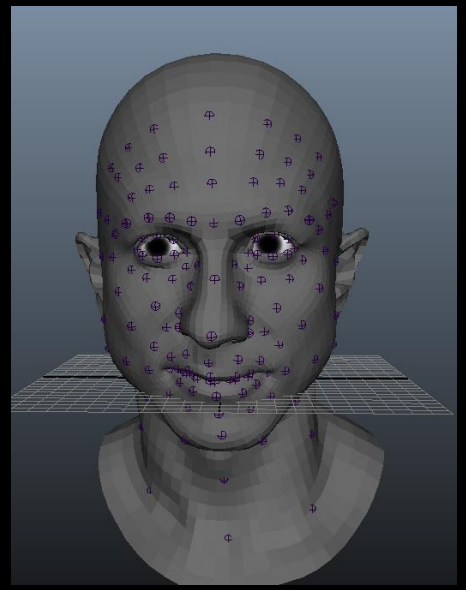
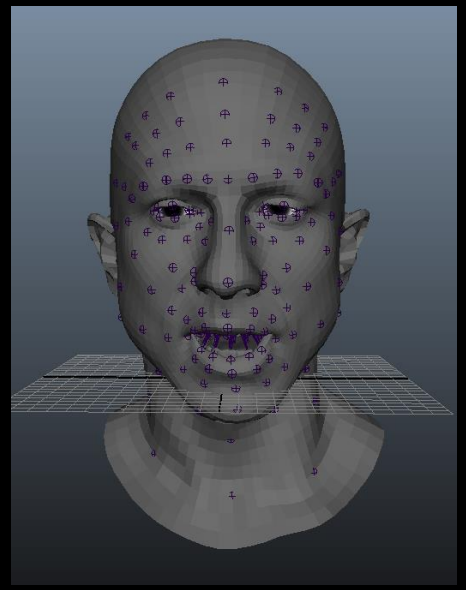
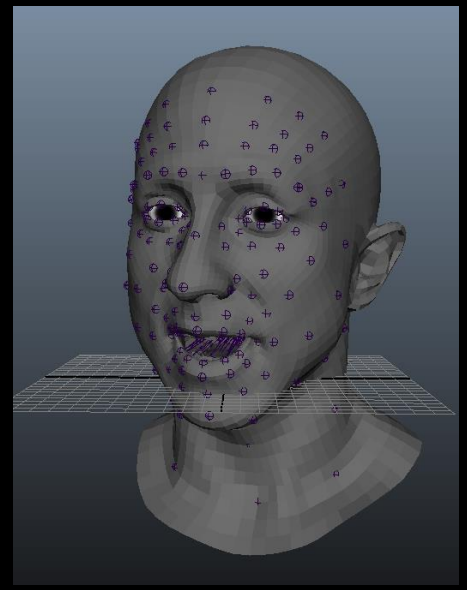
- FACS 24, FACS 33, Compress, Smile, Surprise



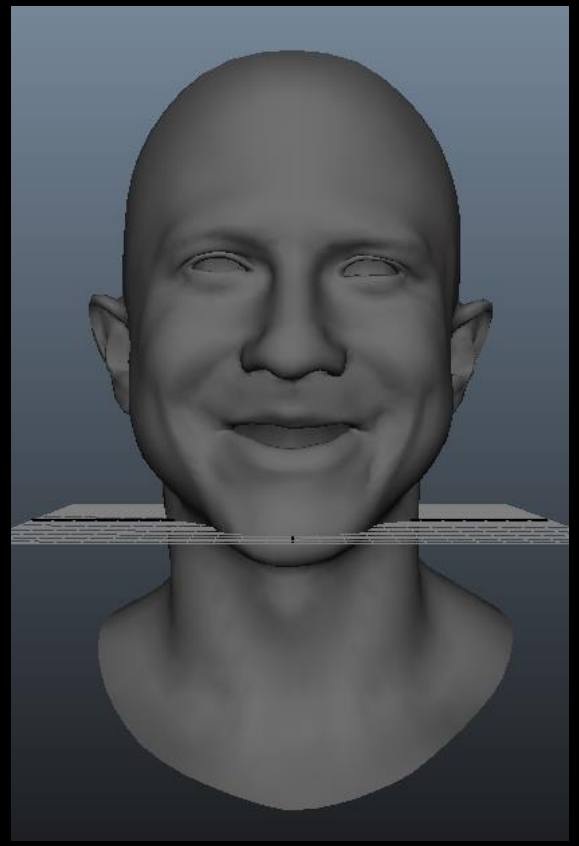
And we have mocap…

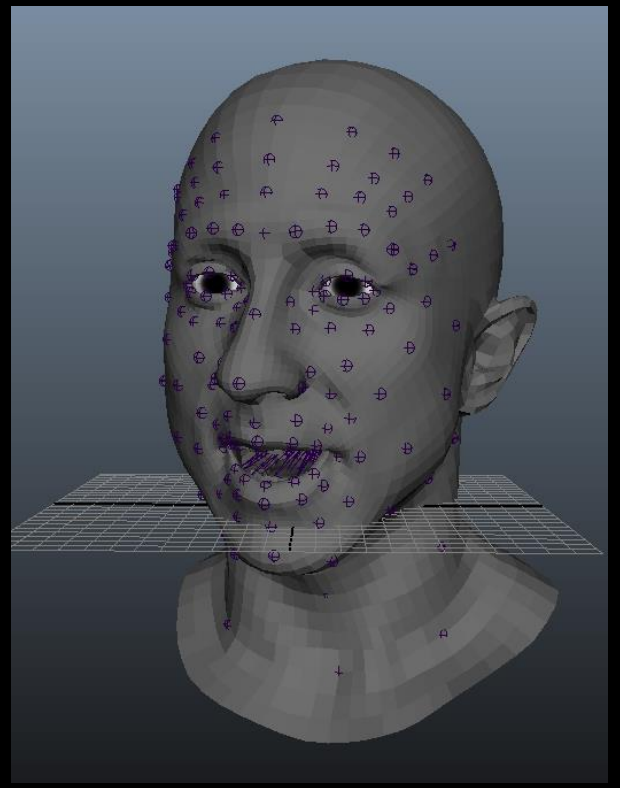
- Now what?

So we have blendshapes…

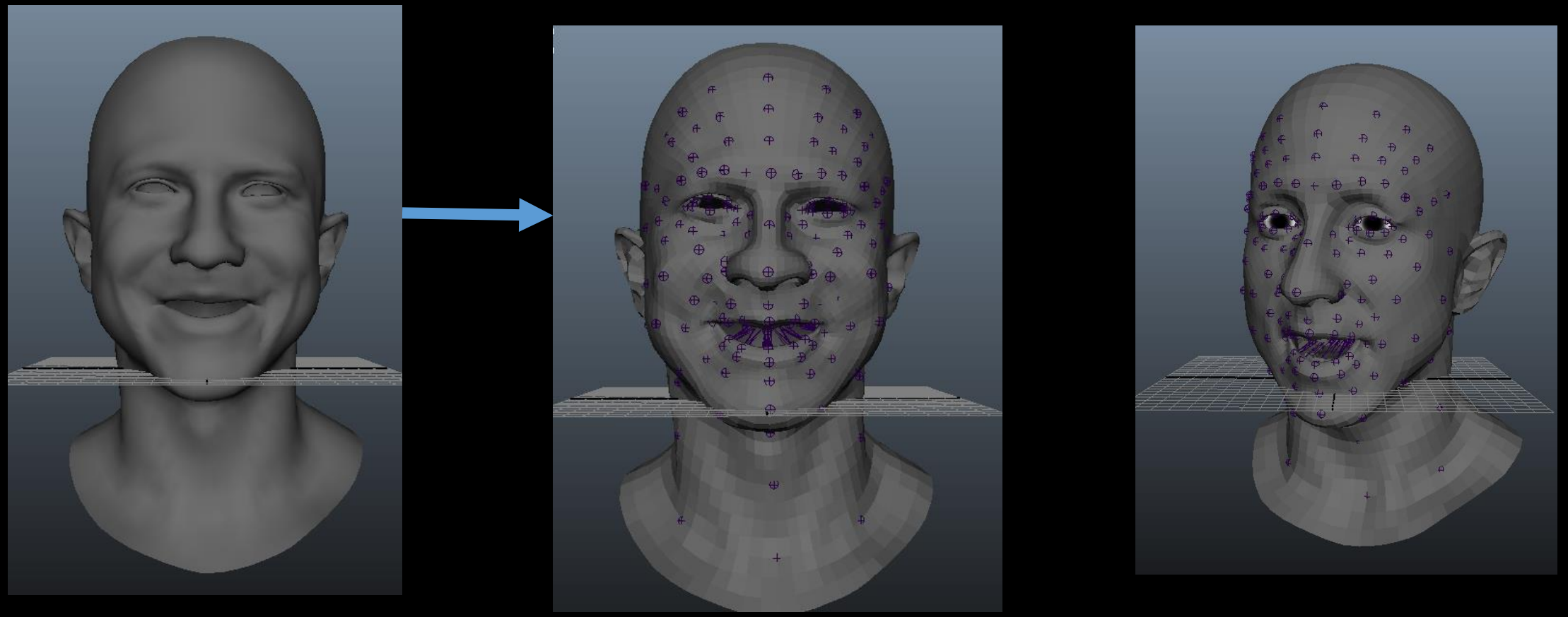
- Blendshapes defined by vertices
- Animation defined by joints





So we have blendshapes…

- Define blendshapes by joints

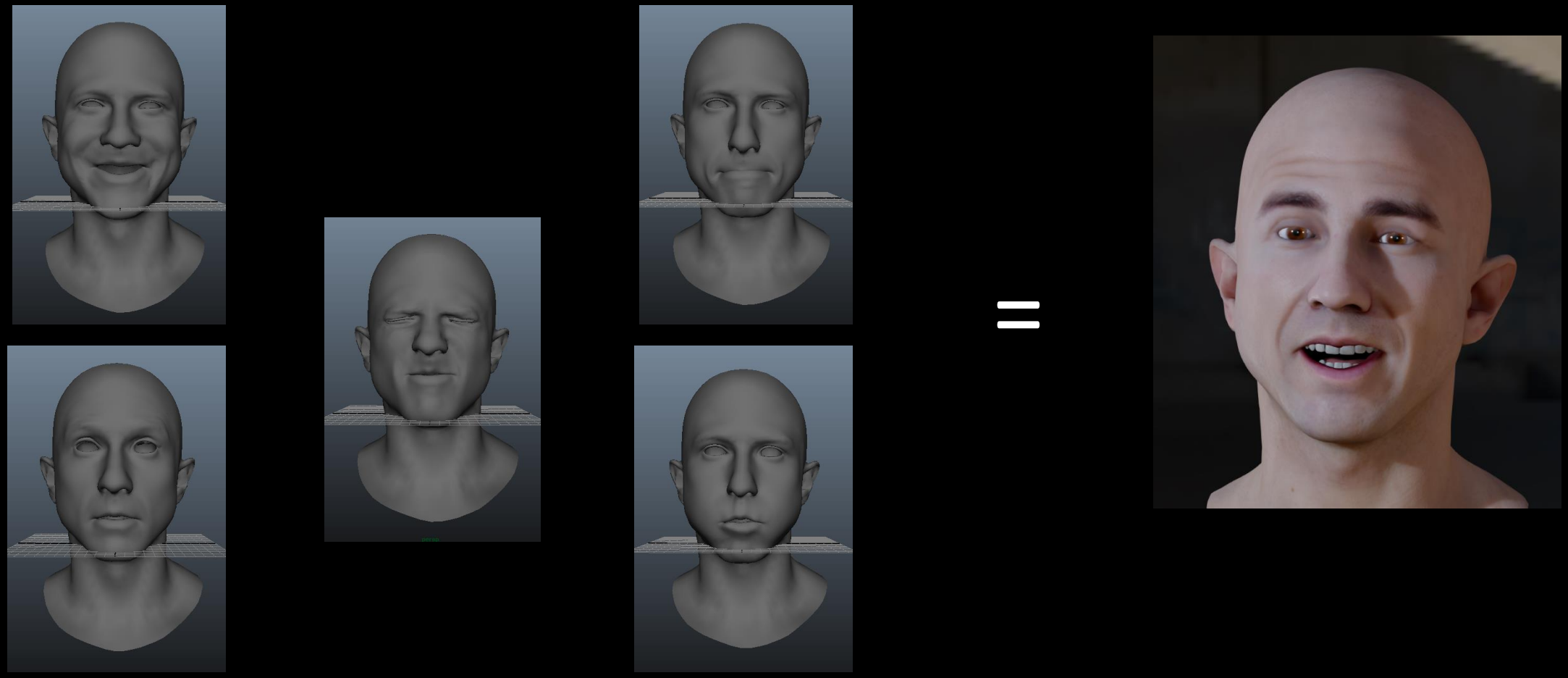


Problem: Correspond Blendshapes

- Find weights for shapes that match joint animation.

Problem: Correspond Blendshapes

- Use those weights for blendshapes.

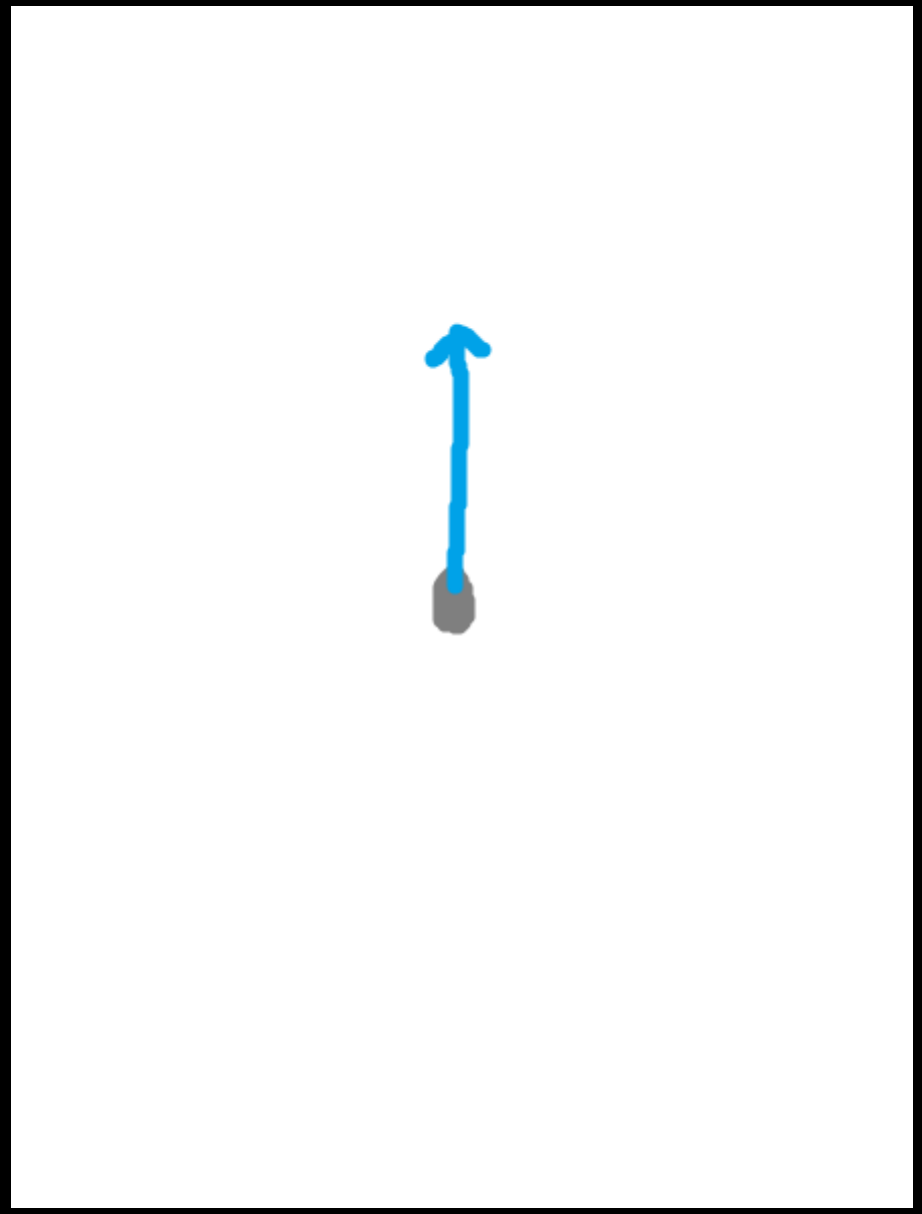


Let’s say we have a joint.

- Current frame animation has offset from neutral
- Find set of shapes to match that offset
- Let’s say we are on the left eyebrow.

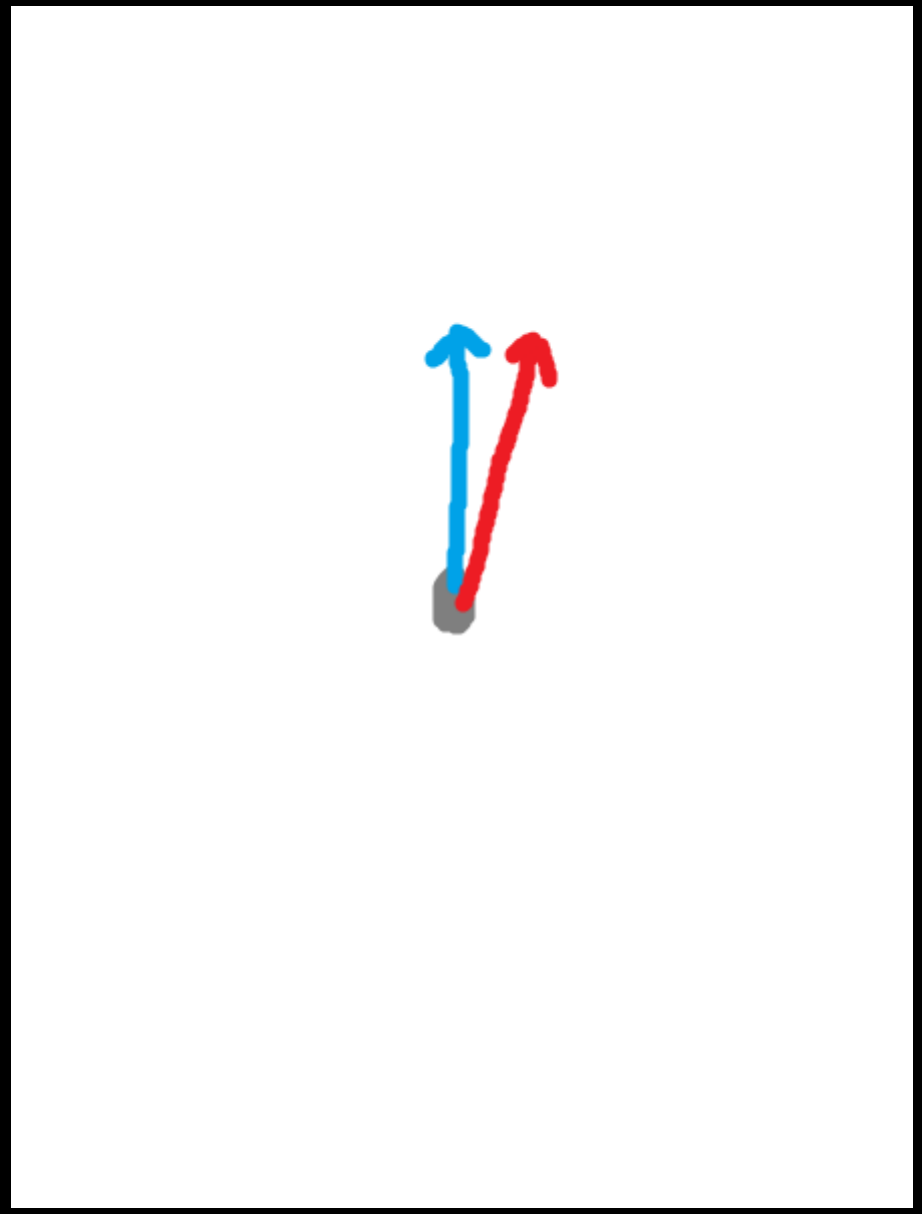
Case 1:

- A_0 = [0,5,0]
- A_1 =
- B =
- $A_0x + A_1y = B$
- x=?, y=?



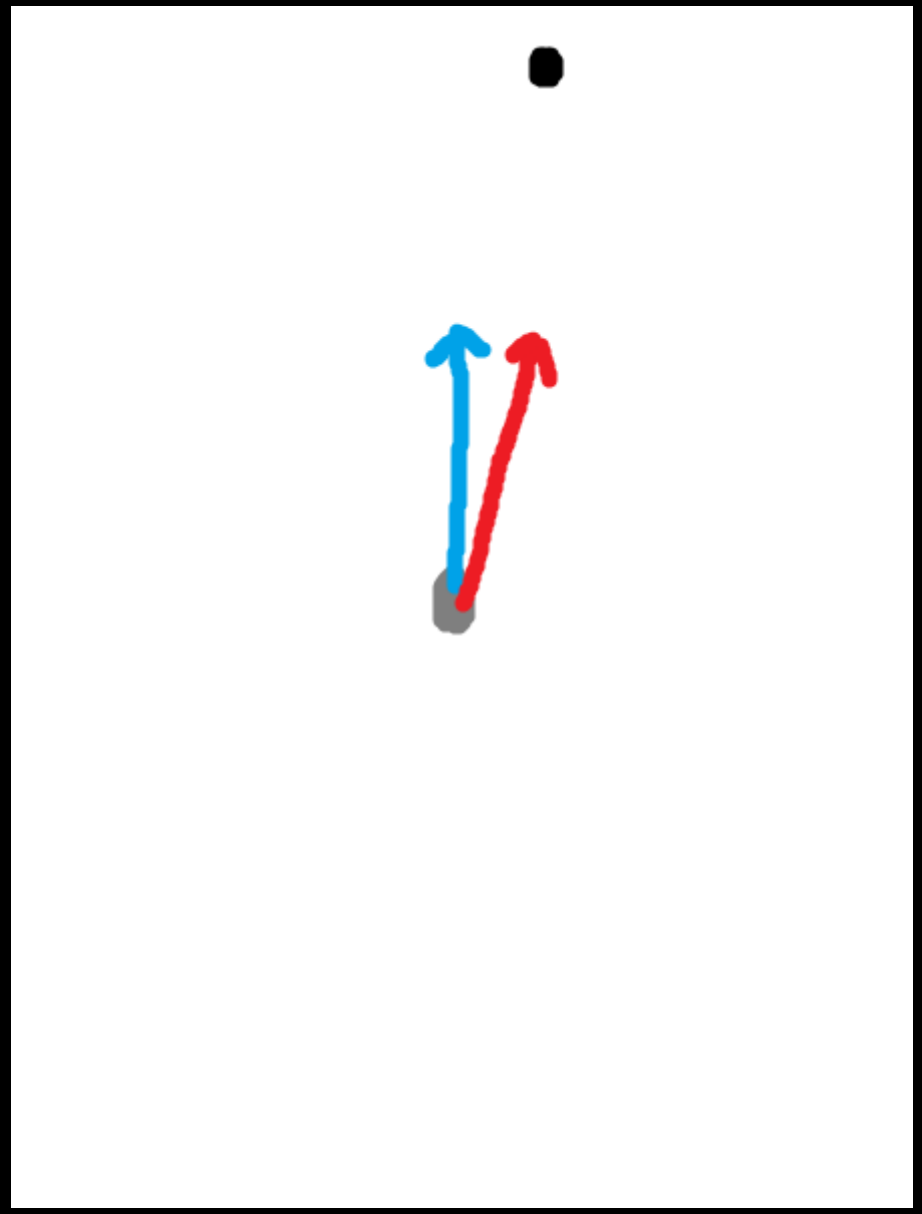
Case 1:

- $A_0 = [0,5,0]$
- $A_1 = [1,5,0]$
- B =
- $A_0x + A_1y = B$
- x=?, y=?



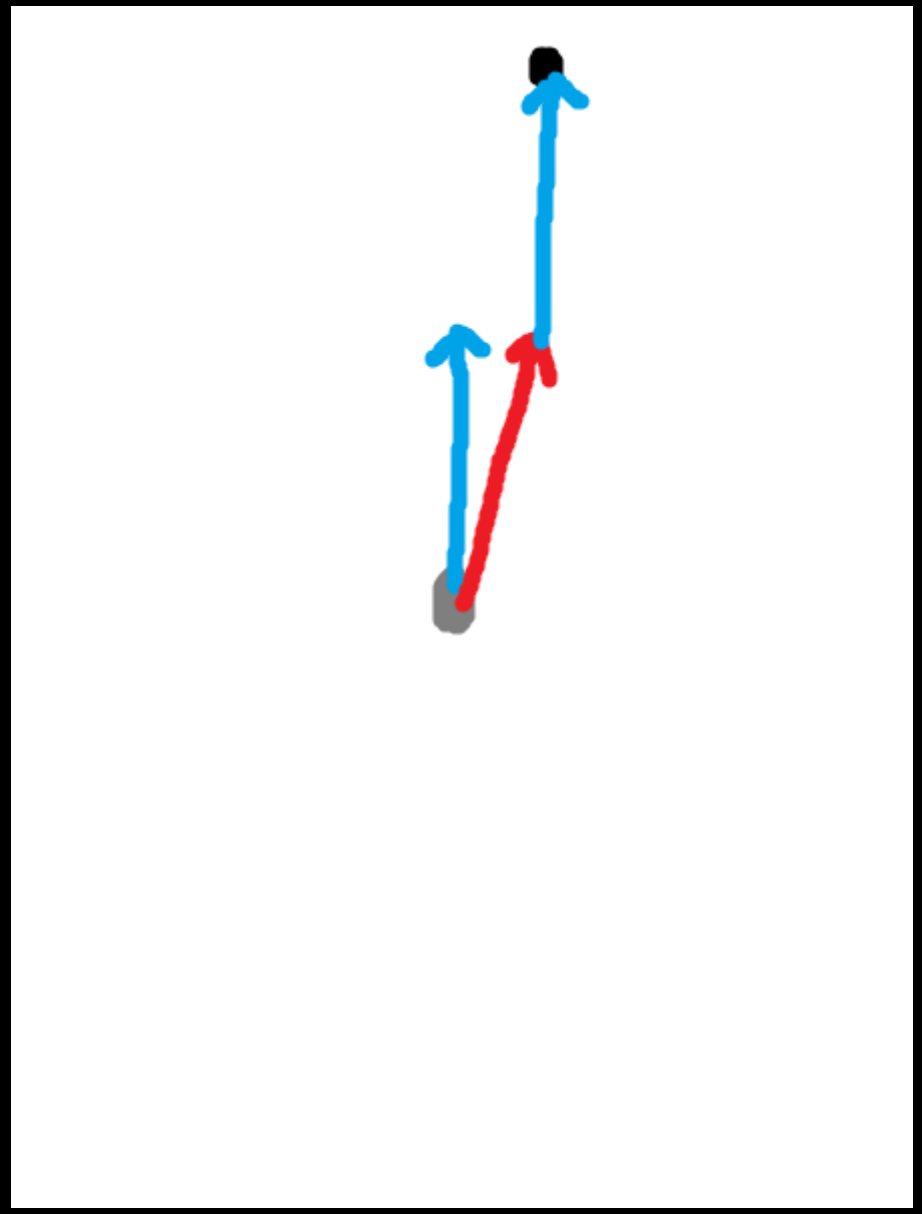
Case 1:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- B = [1,10,0]
- $A_0x + A_1y = B$
- x=?, y=?



Case 1:

- $A_0 = [0,5,0]$
- $A_1 = [1,5,0]$
- $B = [1,10,0]$
- $A_0x + A_1y = B$
- $x=1,\ y=1$

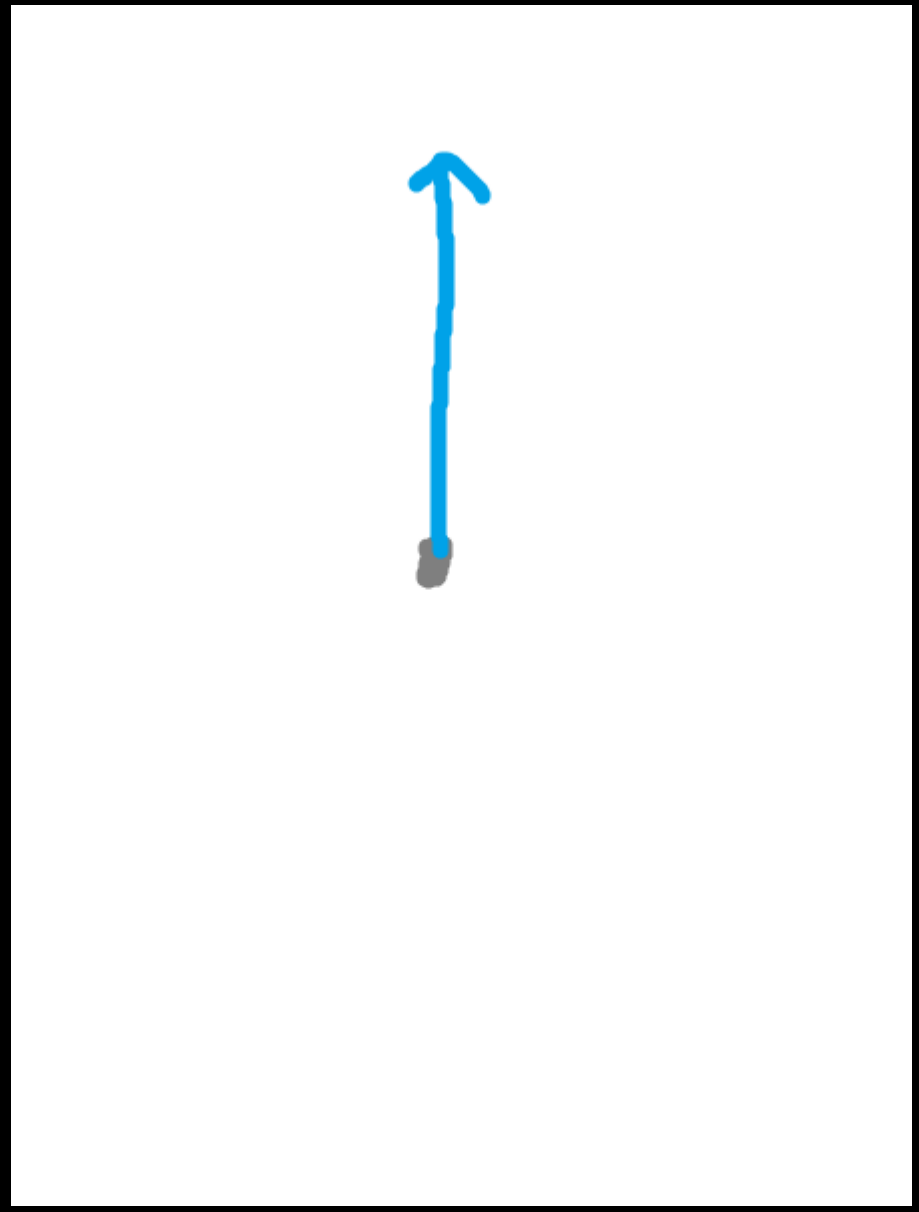


Case 2:

- A_0 =
- A_1 =
- B =
- $A_0x + A_1y = B$
- x=?, y=?

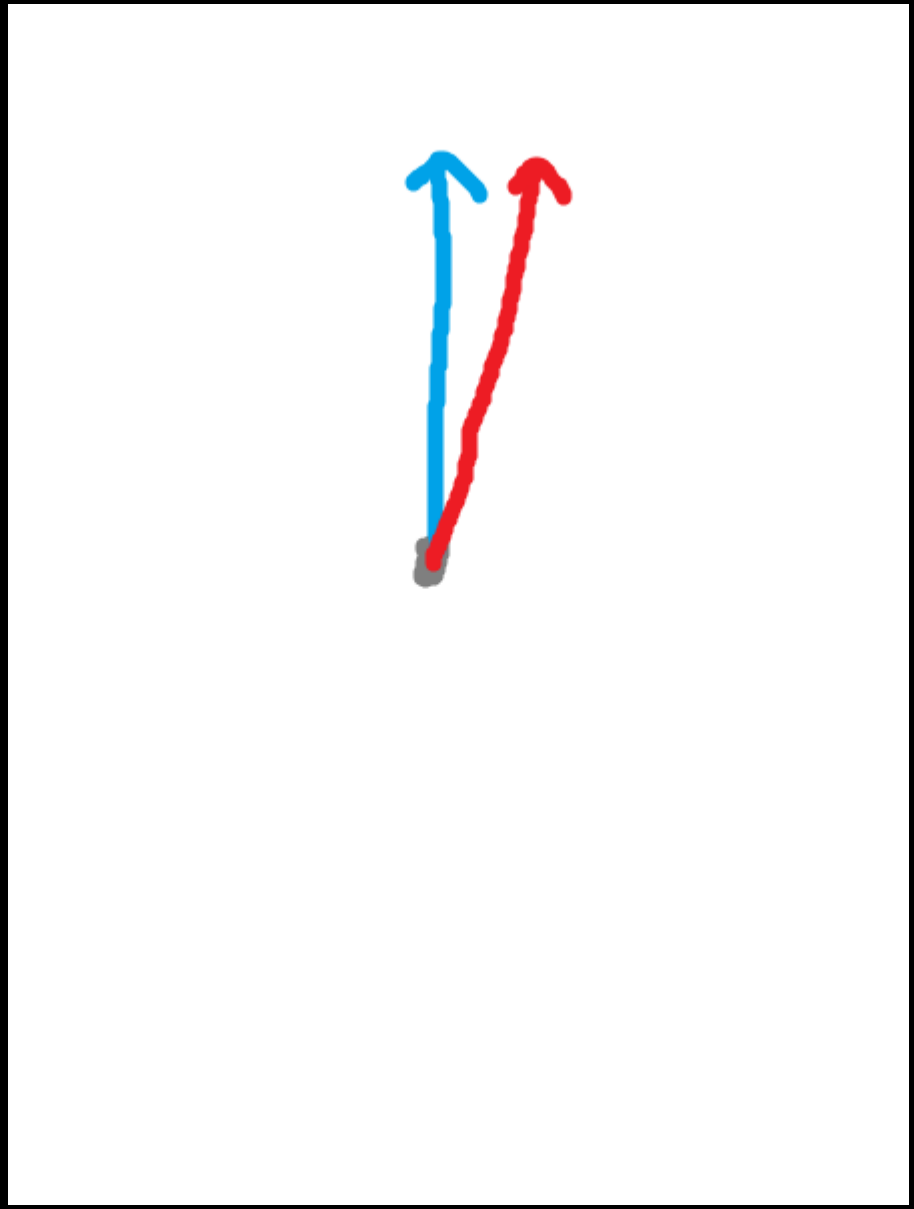
Case 2:

- $A_0 = [0,5,0]$
- $A_1 =$
- $B =$
- $A_0x + A_1y = B$
- x=?, y=?



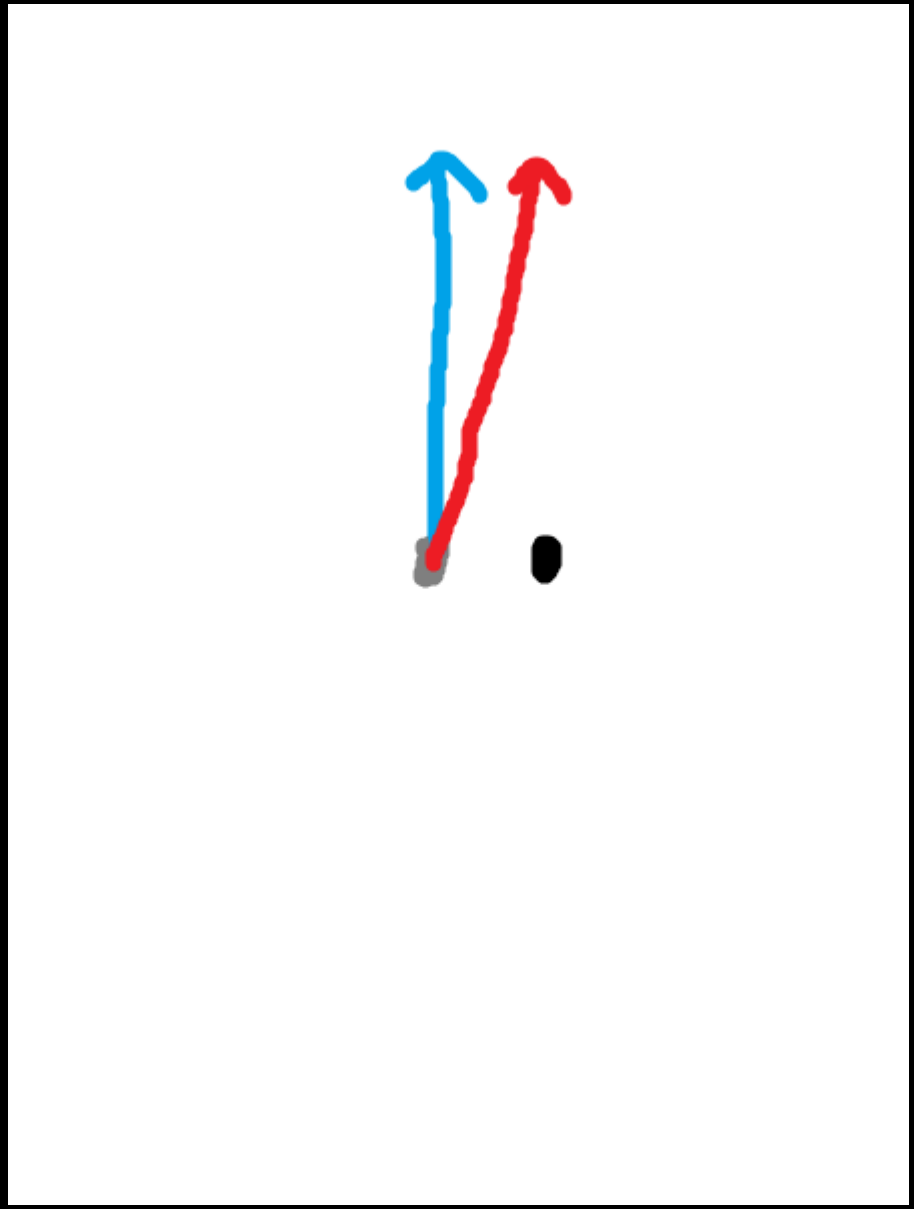
Case 2:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- B =
- $A_0x + A_1y = B$
- x=?, y=?



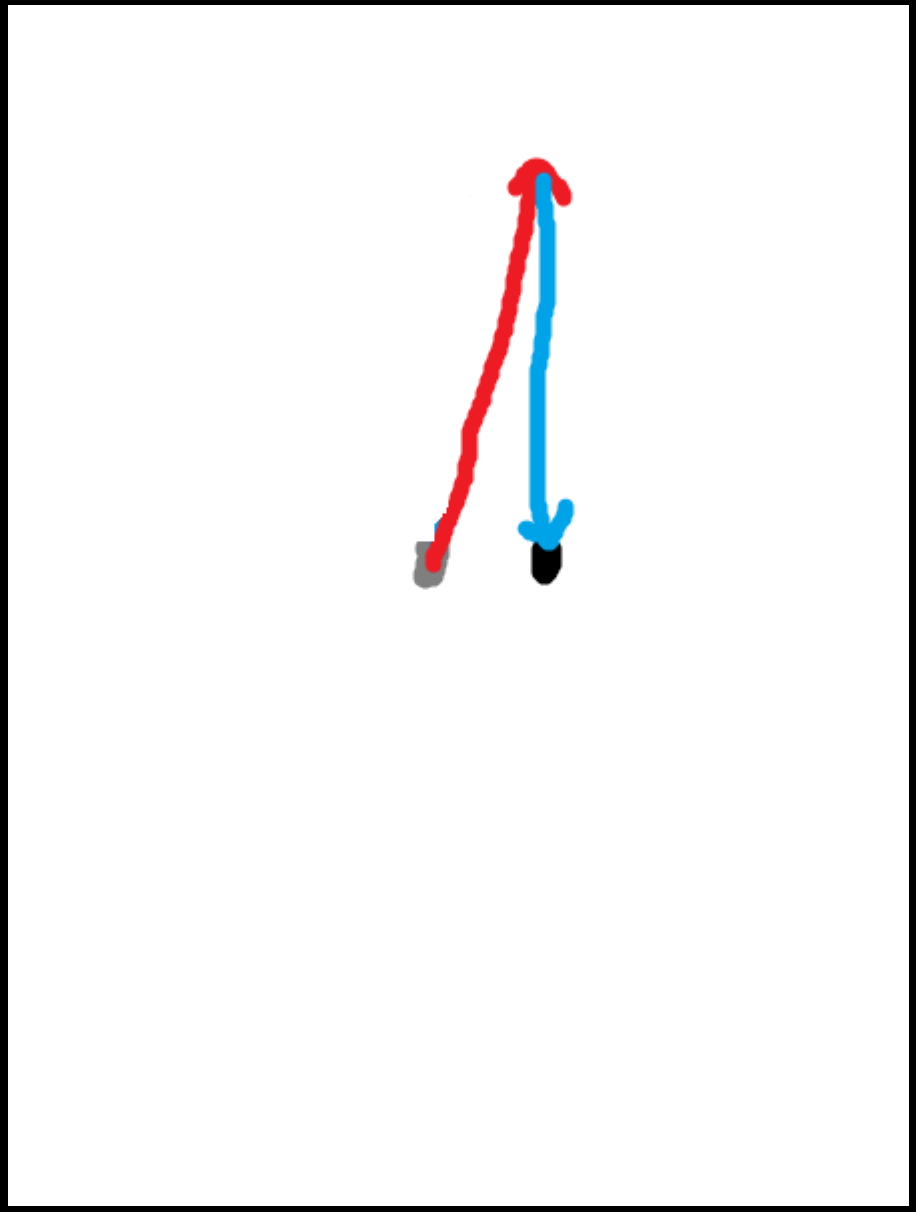
Case 2:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- B = [1,0,0]
- $A_0x + A_1y = B$
- x=?, y=?



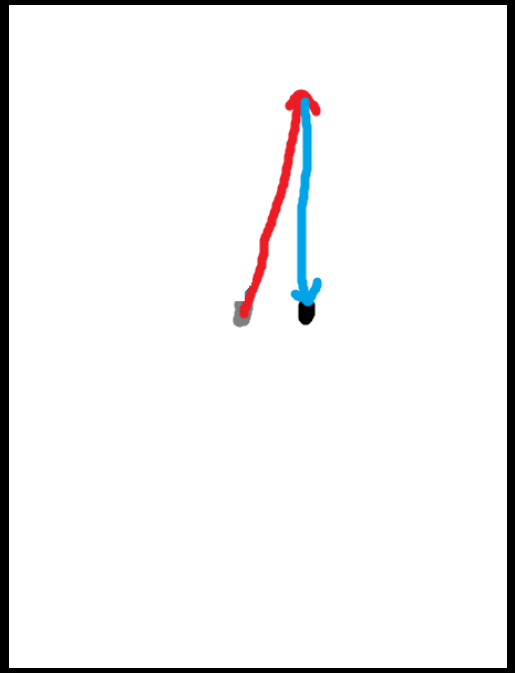
Case 2:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- B = [1,0,0]
- $A_0x + A_1y = B$
- x=-1, y=1



Case 2:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- B = [1,0,0]
- $A_0x + A_1y = B$
- x=-1, y=1



“If at first you don’t succeed, try, try again. Then quit. There’s no point being a damn fool about it.”

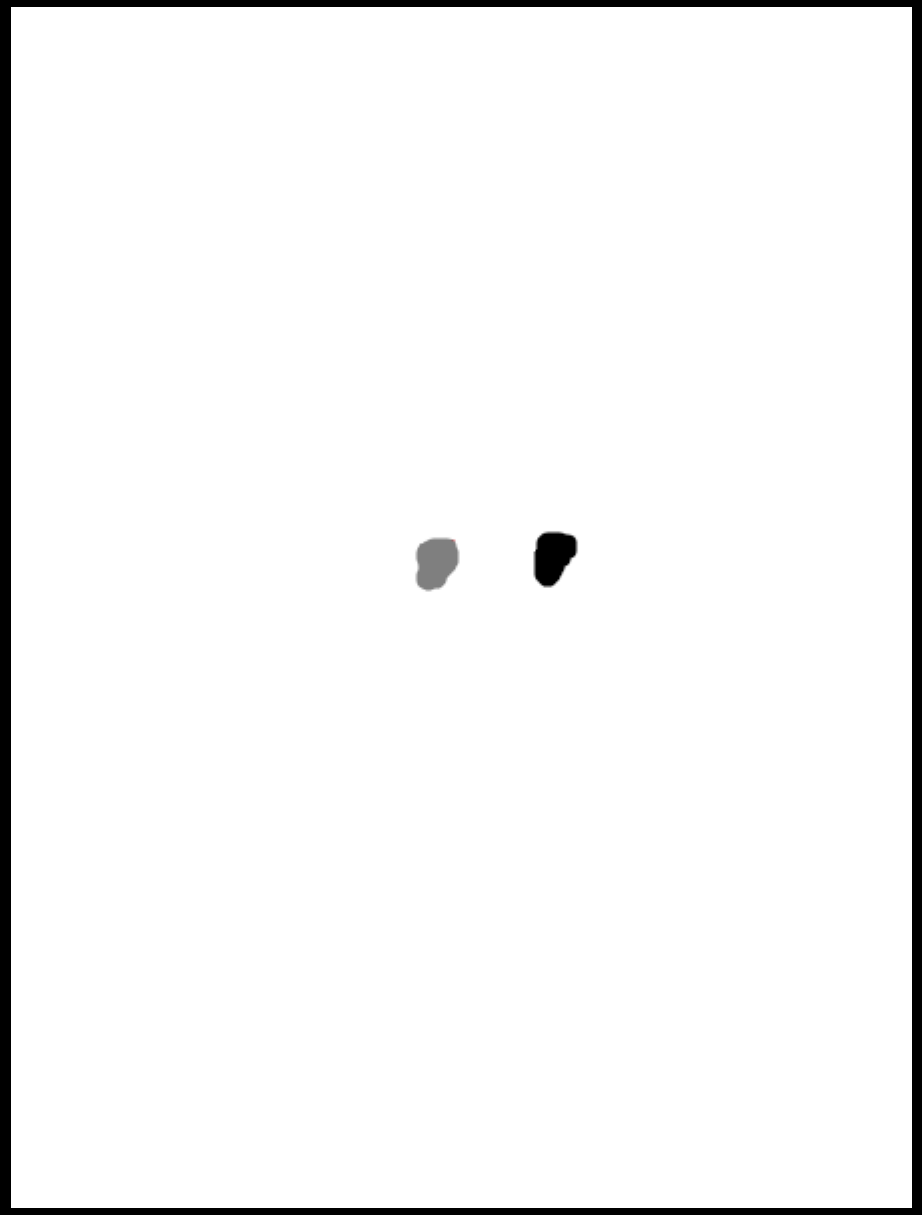
- W. C. Fields

Solution:

- Our matrix is overconstrained
- Disallow negative weights
- Can’t do a “negative” eyebrow up
- Use Non-Negative Least Squares

Case 2:

- $A_0 = [0,5,0]$
- $A_1 = [1,5,0]$
- $B = [1,0,0]$
- $A_0x + A_1y = B$
- $x=0,\ y=0$

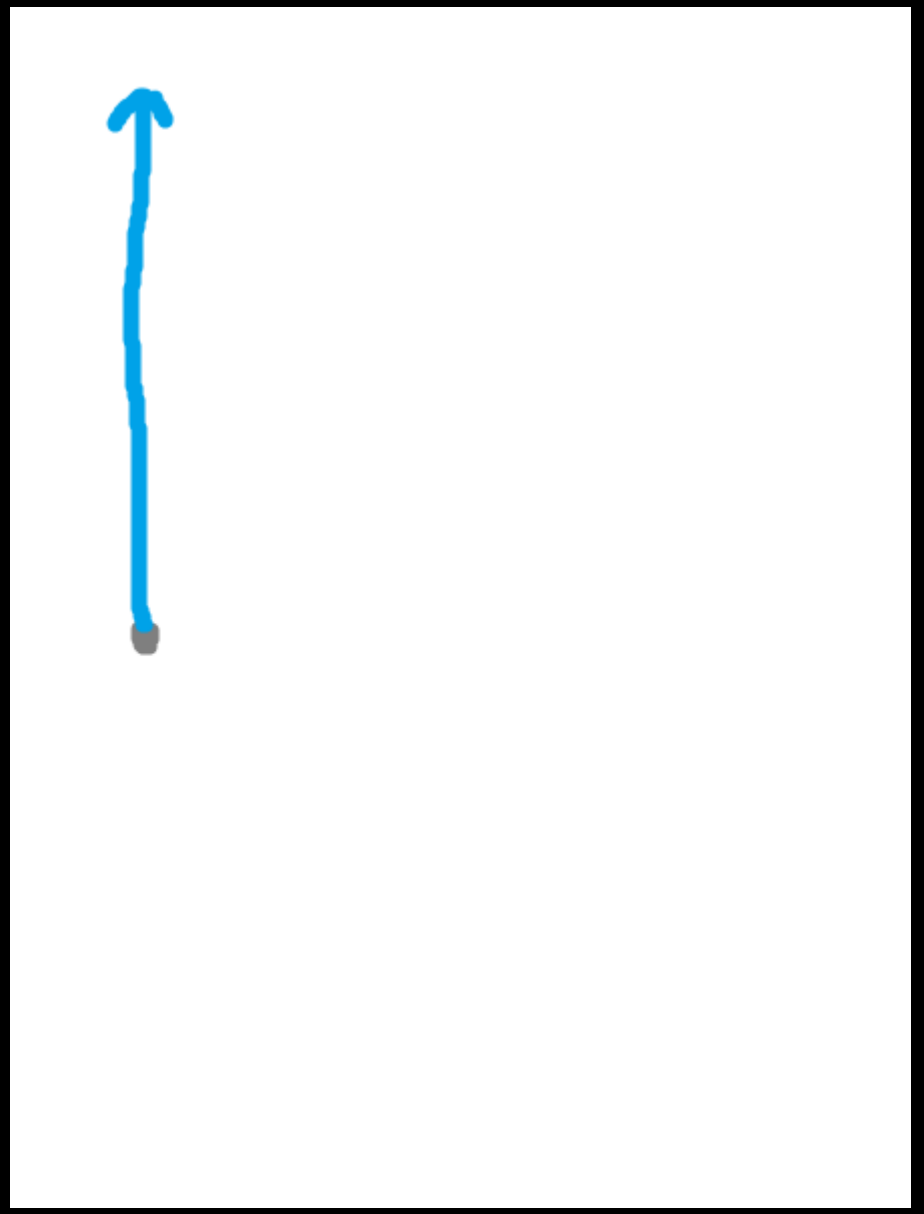


Case 3:

- A_0 =
- A_1 =
- A_2 =
- B =
- $A_0x + A_1y + A_2z = B$
- x=?, y=?, z=?

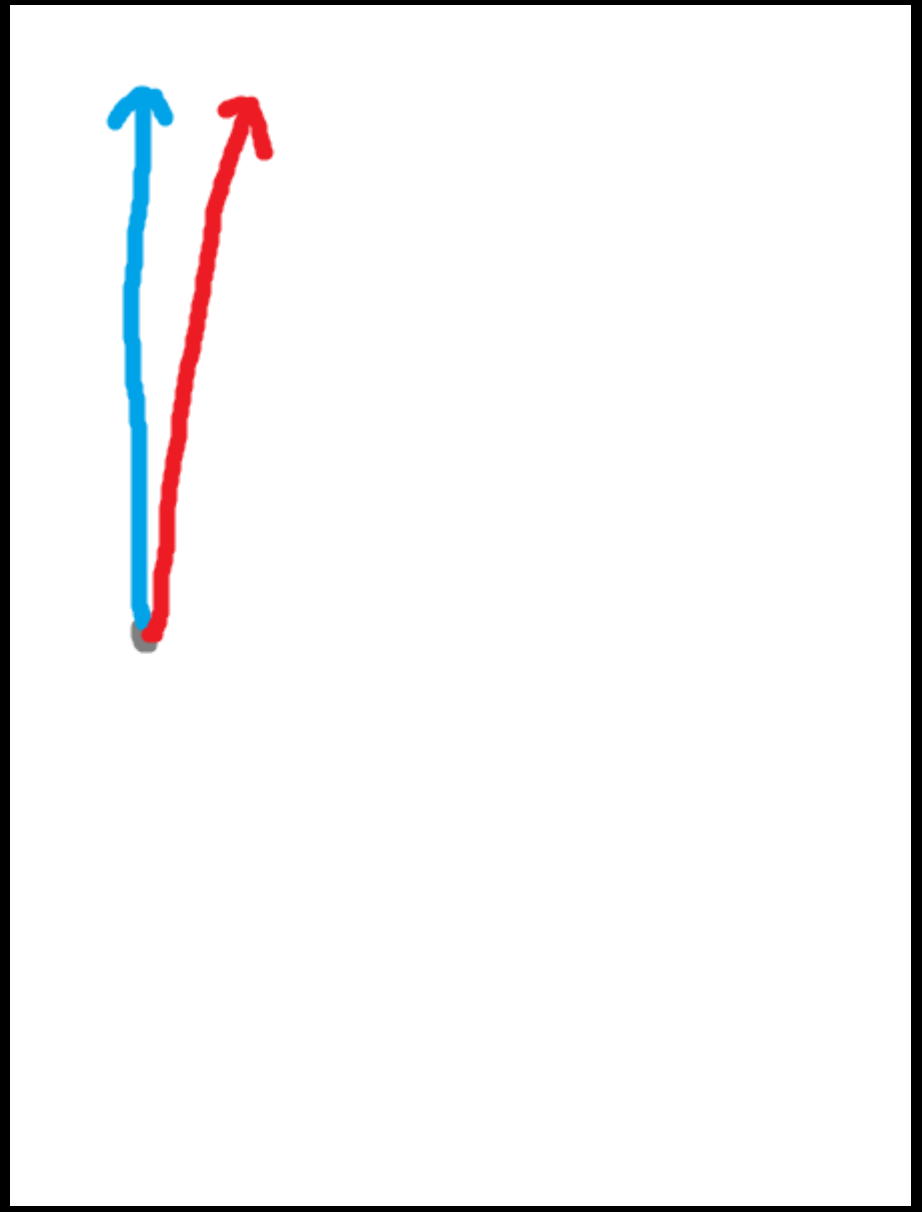
Case 3:

- A_0 = [0,5,0]
- A_1 =
- A_2 =
- B =
- $A_0x + A_1y + A_2z = B$
- x=?, y=?, z=?



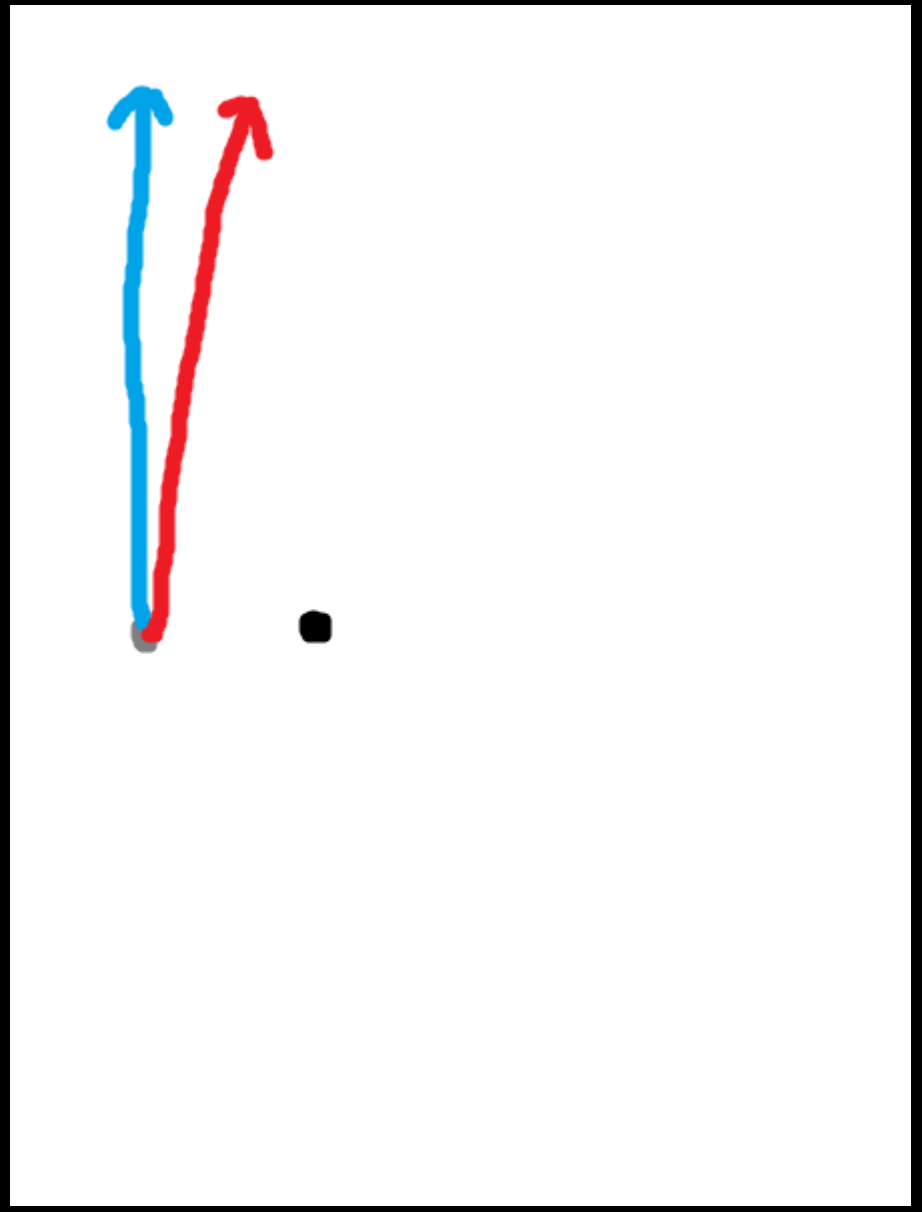
Case 3:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- A_2 =
- B =
- $A_0x + A_1y + A_2z = B$
- x=?, y=?, z=?



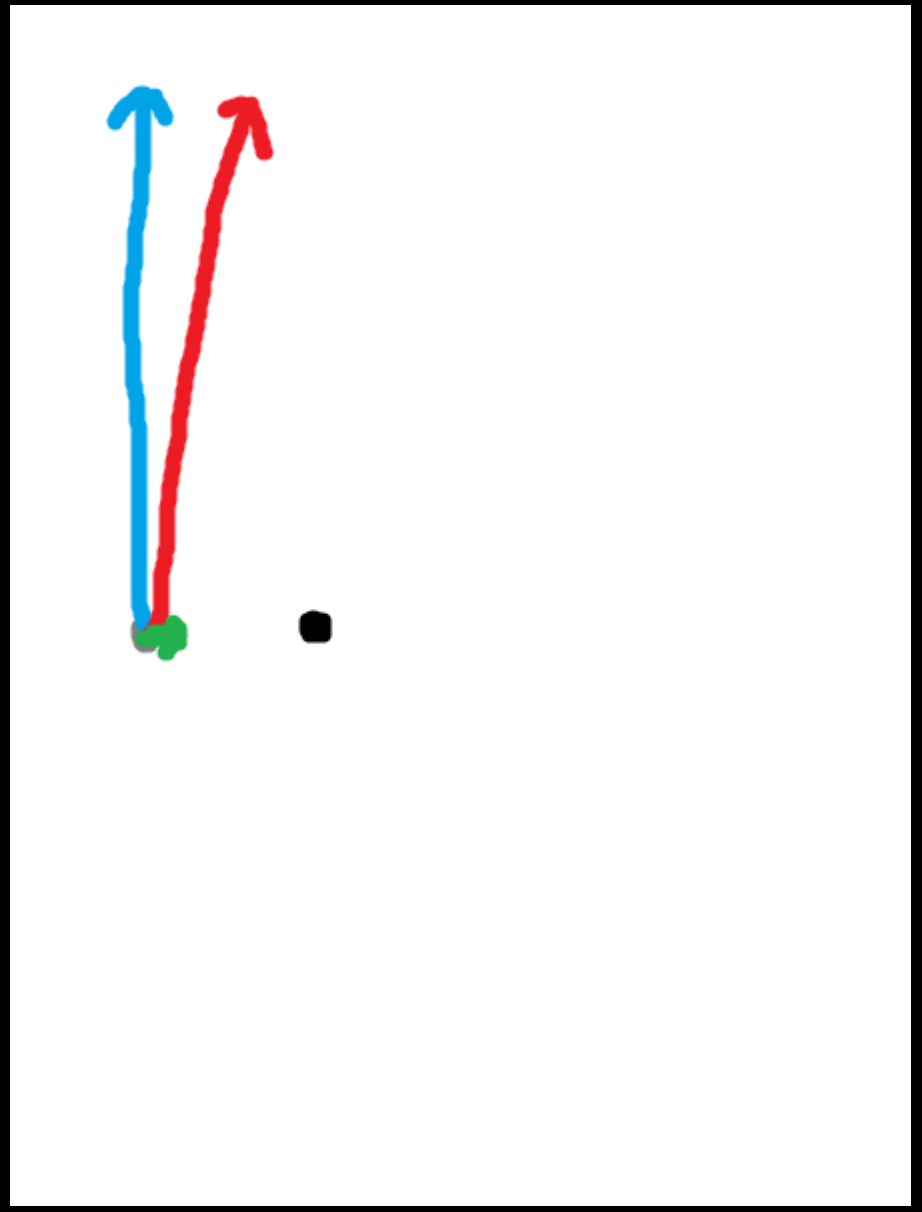
Case 3:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- A_2 =
- B = [1,0,0]
- $A_0x + A_1y + A_2z = B$
- x=?, y=?, z=?



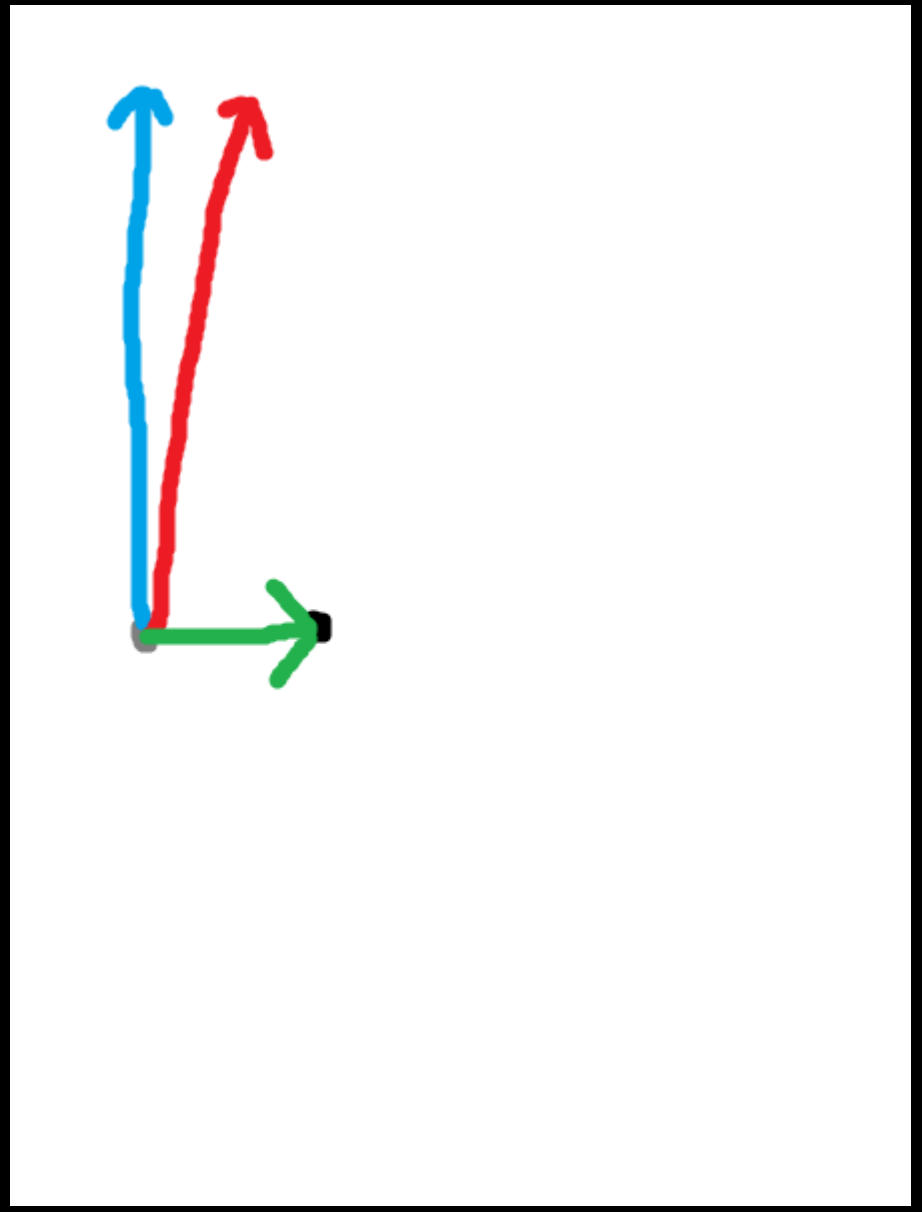
Case 3:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- A_2 = [.1,0,0]
- B = [1,0,0]
- $A_0x + A_1y + A_2z = B$
- x=?, y=?, z=?



Case 3:

- $A_0 = [0,5,0]$
- $A_1 = [1,5,0]$
- $A_2 = [.1,0,0]$
- $B = [1,0,0]$
- $A_0x + A_1y + A_2z = B$
- $x=0,\ y=0,\ z=10$

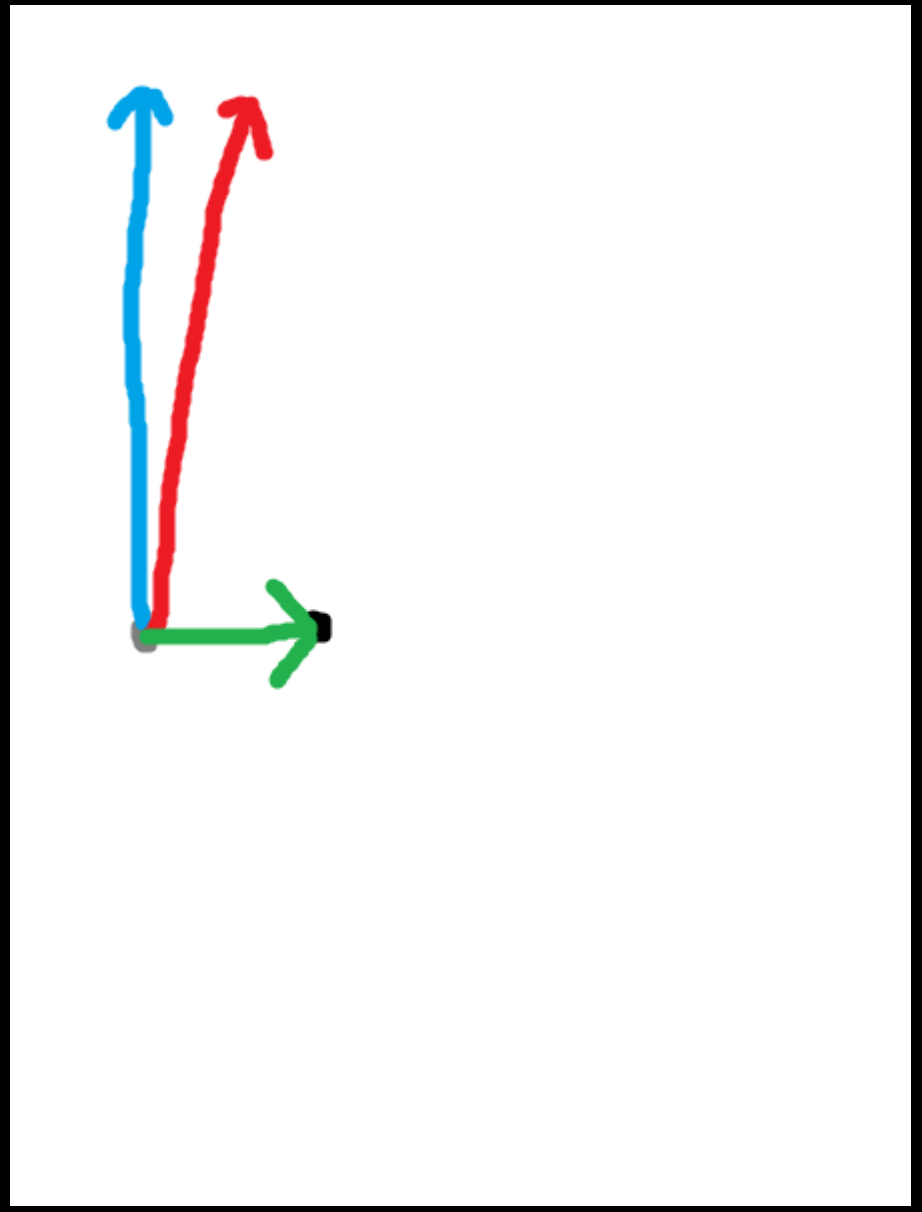


Solution:

- Disallow weights greater than 1.0
  - Might go a little higher in some cases (like 1.5)
- Could use a proper range solver
- I just clamped at the end
  - Somewhat rare case

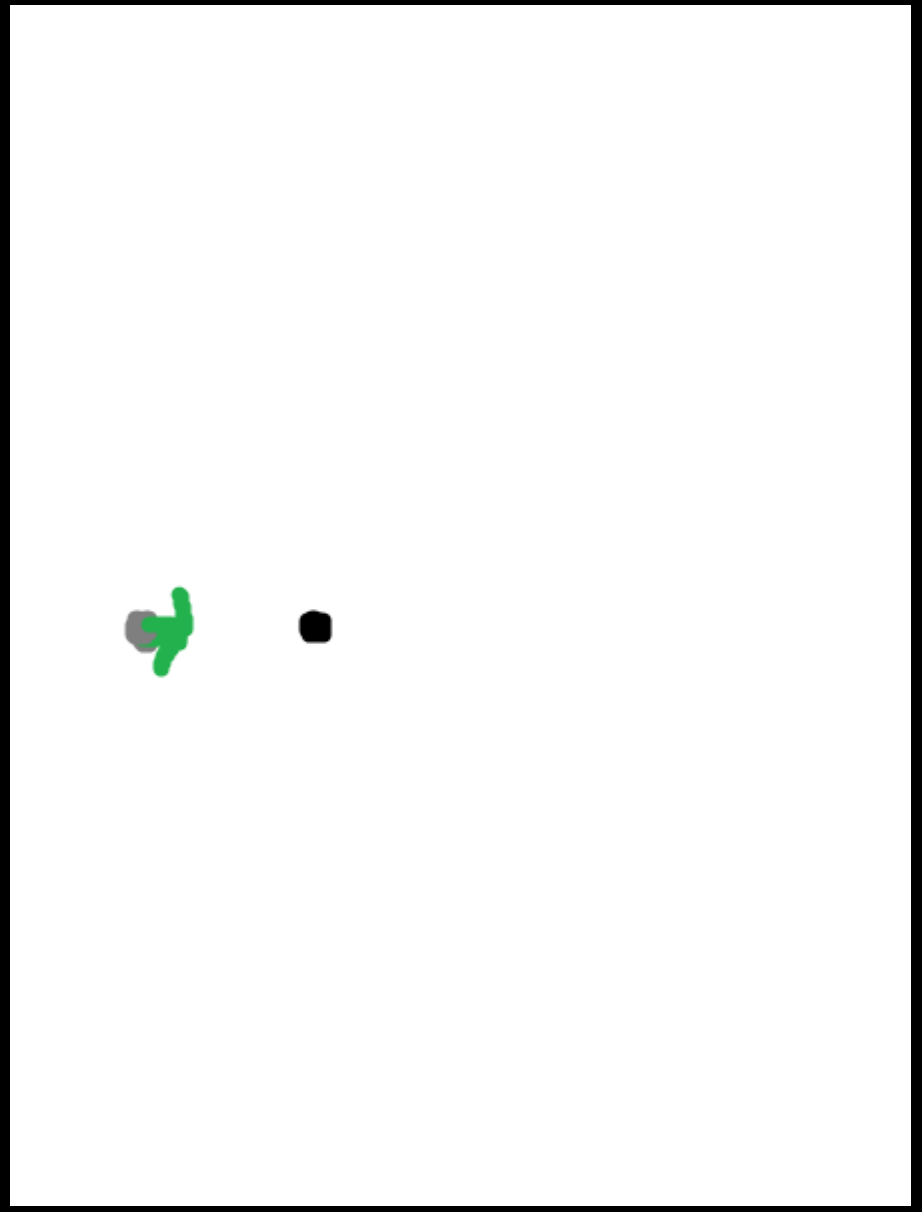
Case 3:

- $A_0 = [0,5,0]$
- $A_1 = [1,5,0]$
- $A_2 = [.1,0,0]$
- $B = [1,0,0]$
- $A_0x + A_1y + A_2z = B$
- $x=0,\ y=0,\ z=10$



Case 3:

- A_0 = [0,5,0]
- A_1 = [1,5,0]
- A_2 = [.1,0,0]
- B = [1,0,0]
- $A_0x + A_1y + A_2z = B$
- x=0, y=0, z=1

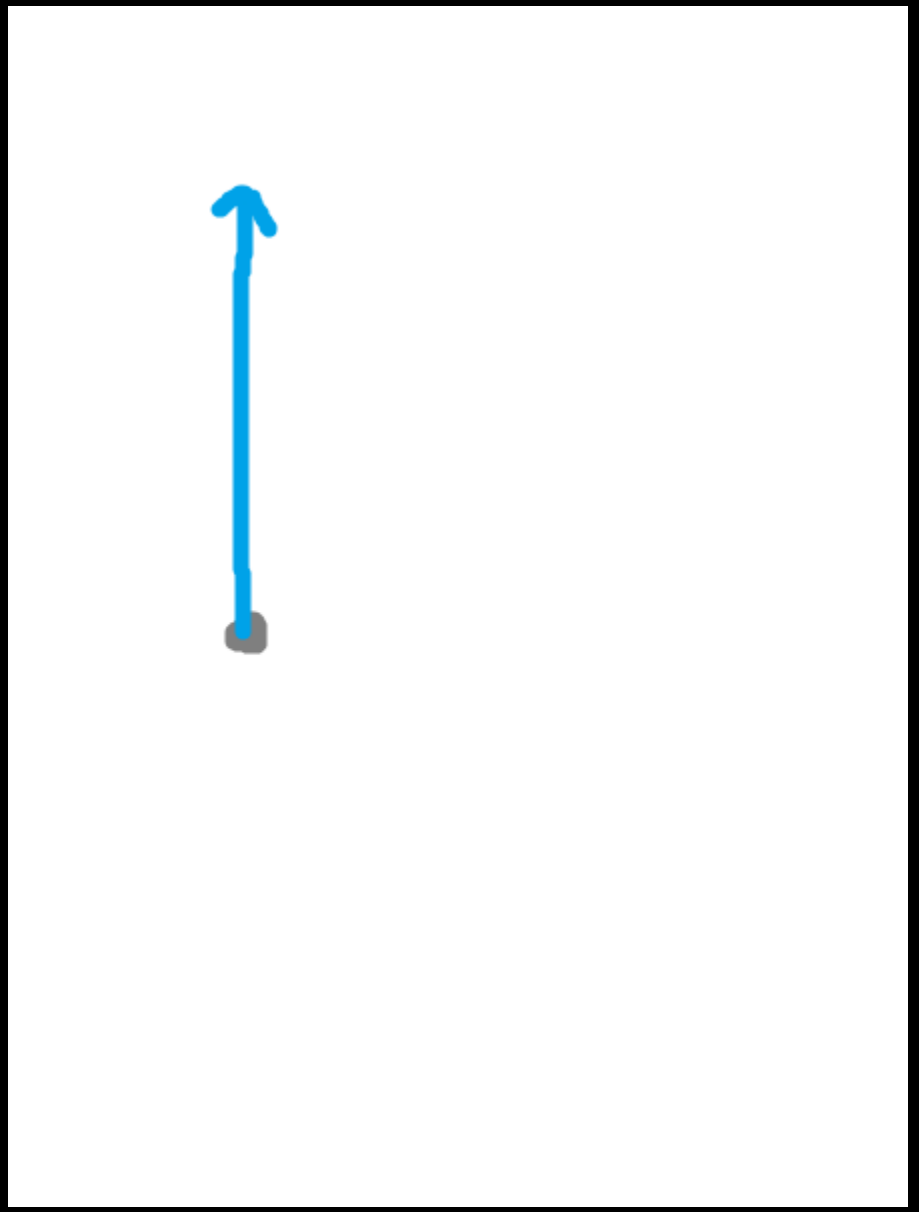


Case 4:

- A_0 =
- A_1 =
- B =
- $A_0x + A_1y = B$
- x=?, y=?

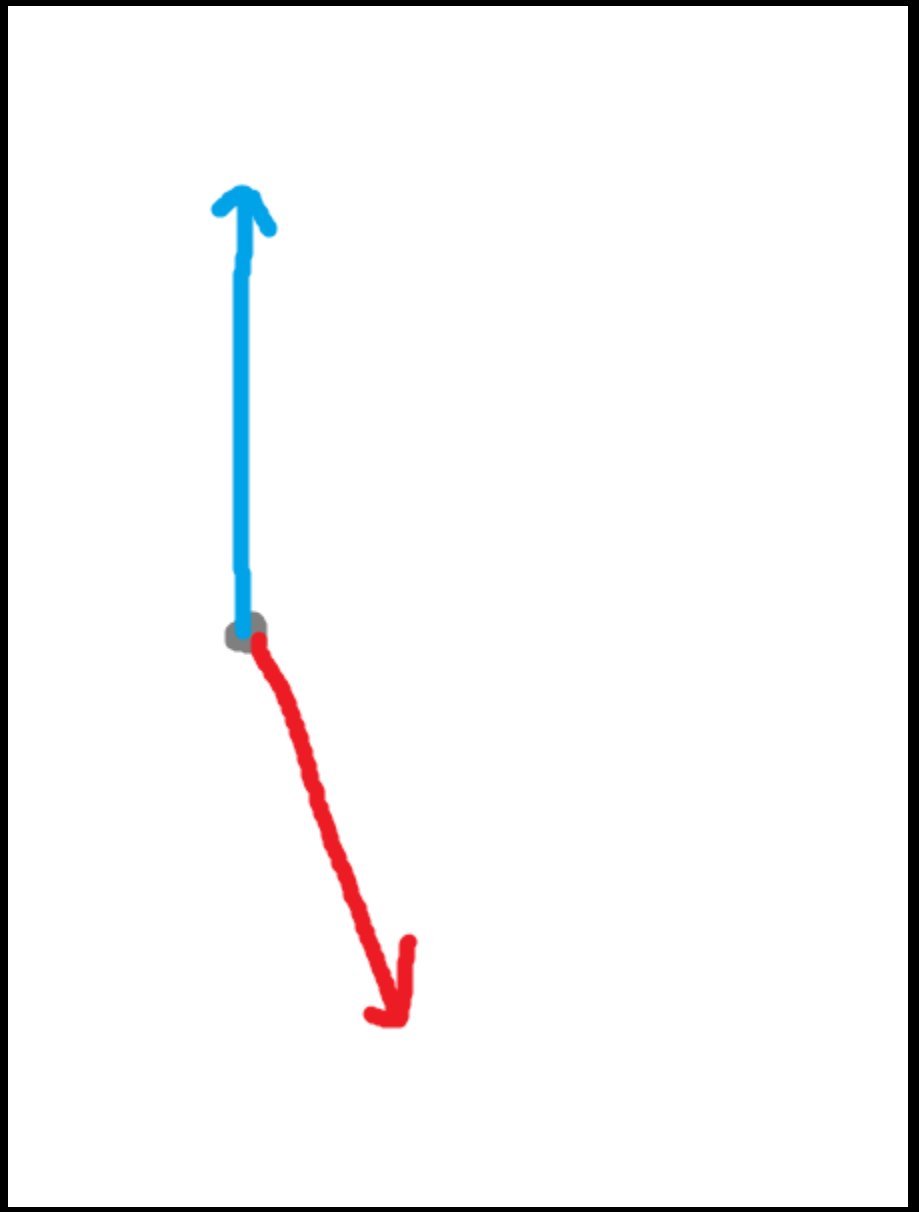
Case 4:

- $A_0 = [0,5,0]$
- $A_1 =$
- $B =$
- $A_0x + A_1y = B$
- x=?, y=?



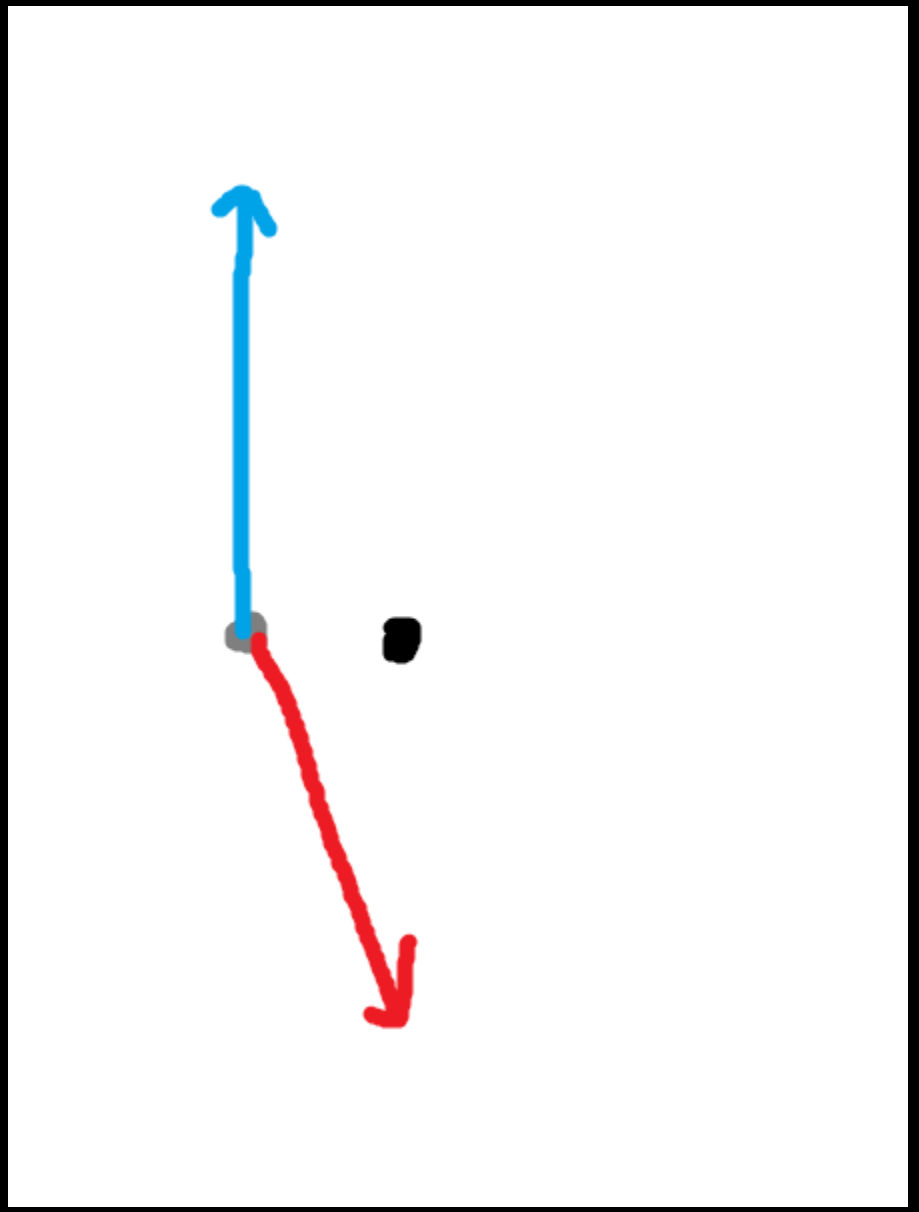
Case 4:

- A_0 = [0,5,0]
- A_1 = [1,-5,0]
- B =
- $A_0x + A_1y = B$
- x=?, y=?



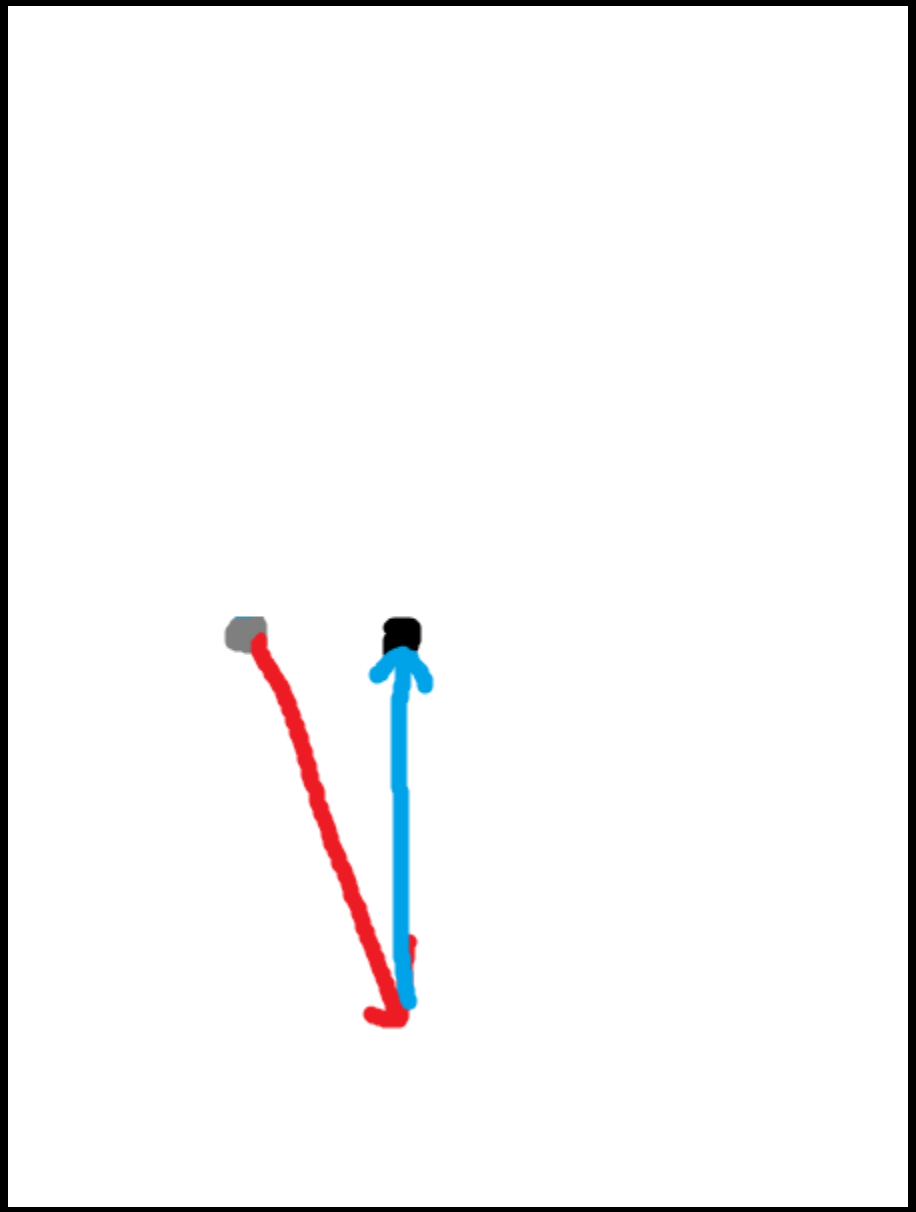
Case 4:

- $A_0 = [0,5,0]$
- $A_1 = [1,-5,0]$
- $B = [1,0,0]$
- $A_0x + A_1y = B$
- x=?, y=?



Case 4:

- A_0 = [0,5,0]
- A_1 = [1,-5,0]
- B = [1,0,0]
- $A_0x + A_1y = B$
- x=1, y=1



Usual Solution:

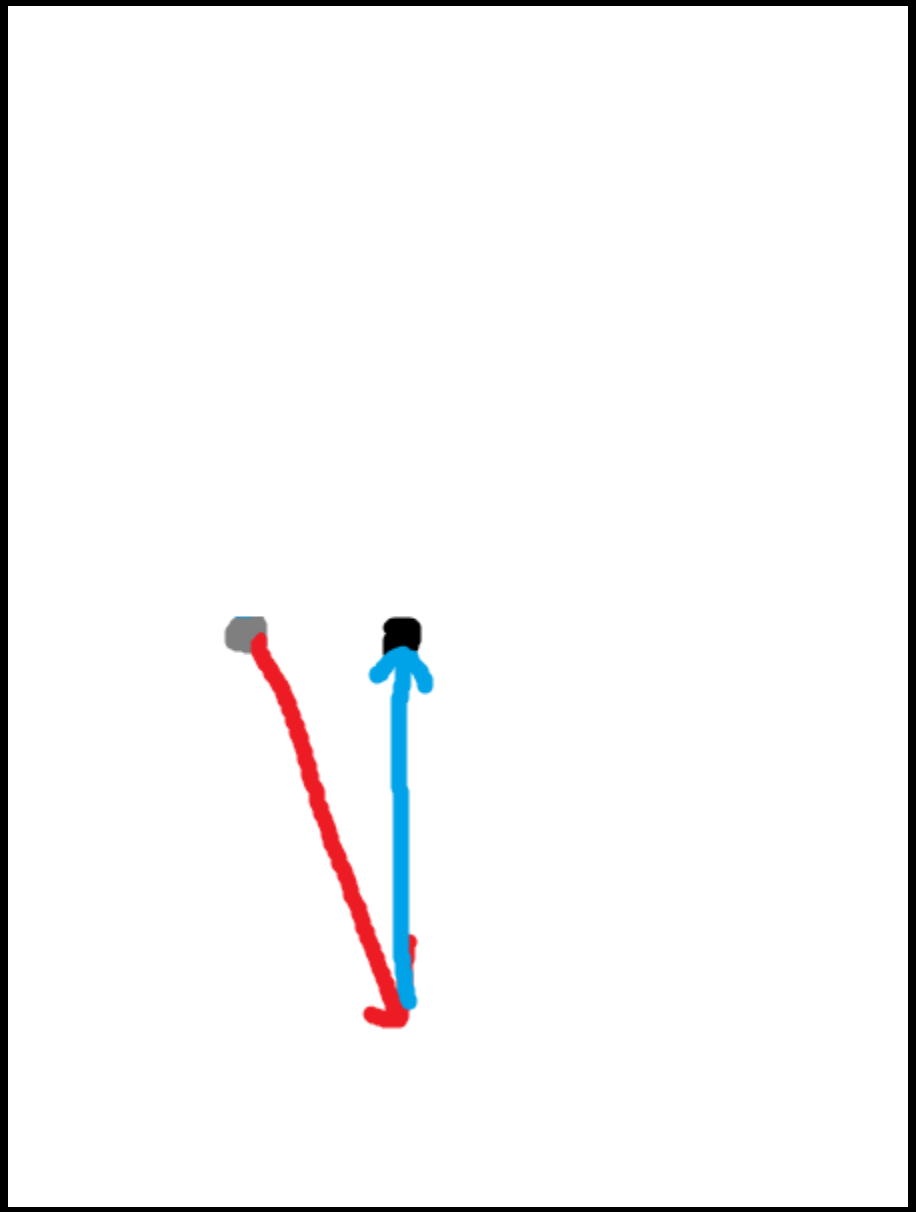
- Disallow certain blendshape combinations
  - Can’t have brow up and brow down
- Lot’s of combinations to worry about
- Did not do this.

Solution:

- Change the joints data
- Instead of 3 vector [x,y,z]
- Make it 6 vector [+x,-x,+y,-y,+z,-z]

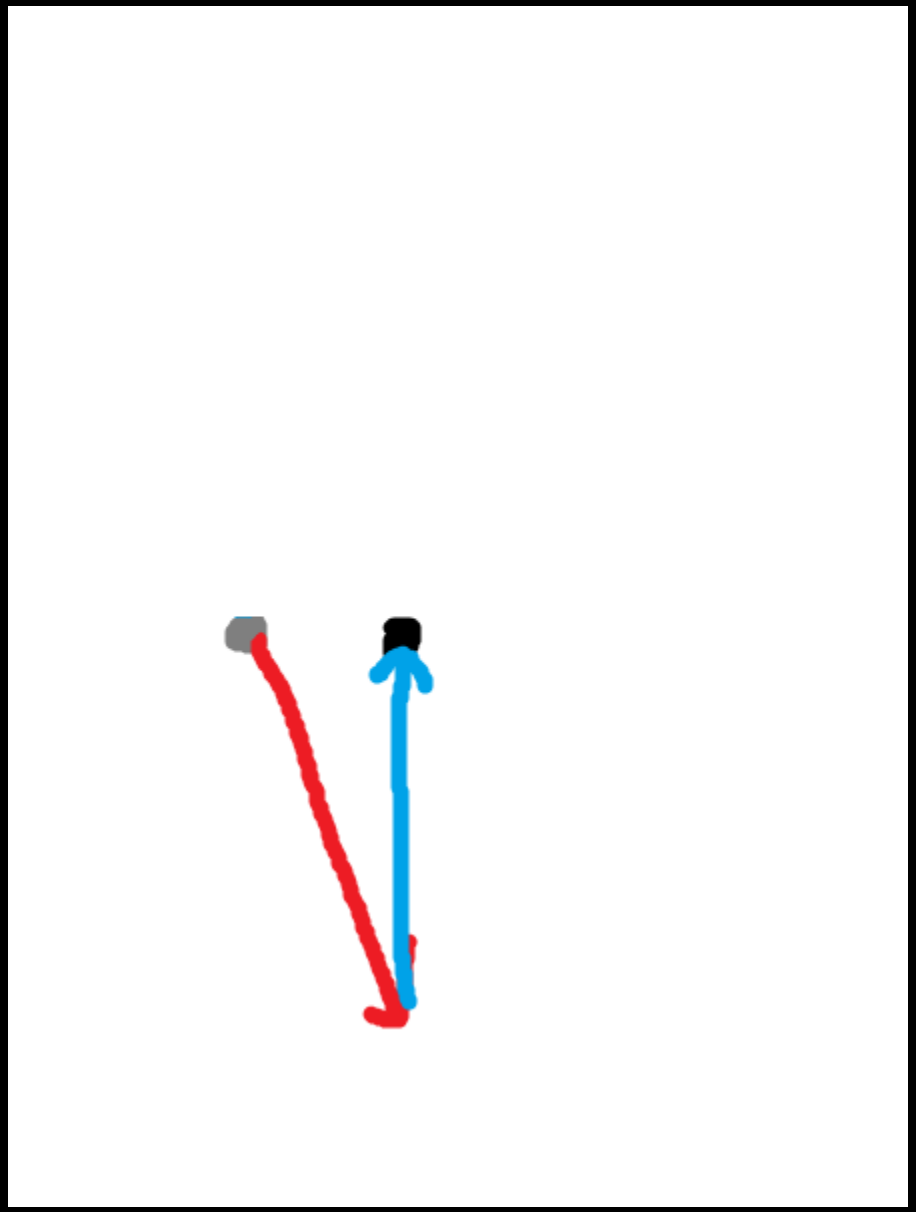
Case 4:

- A_0 = [0,5,0]
- A_1 = [1,-5,0]
- B = [1,0,0]
- $A_0x + A_1y = B$
- x=1, y=1



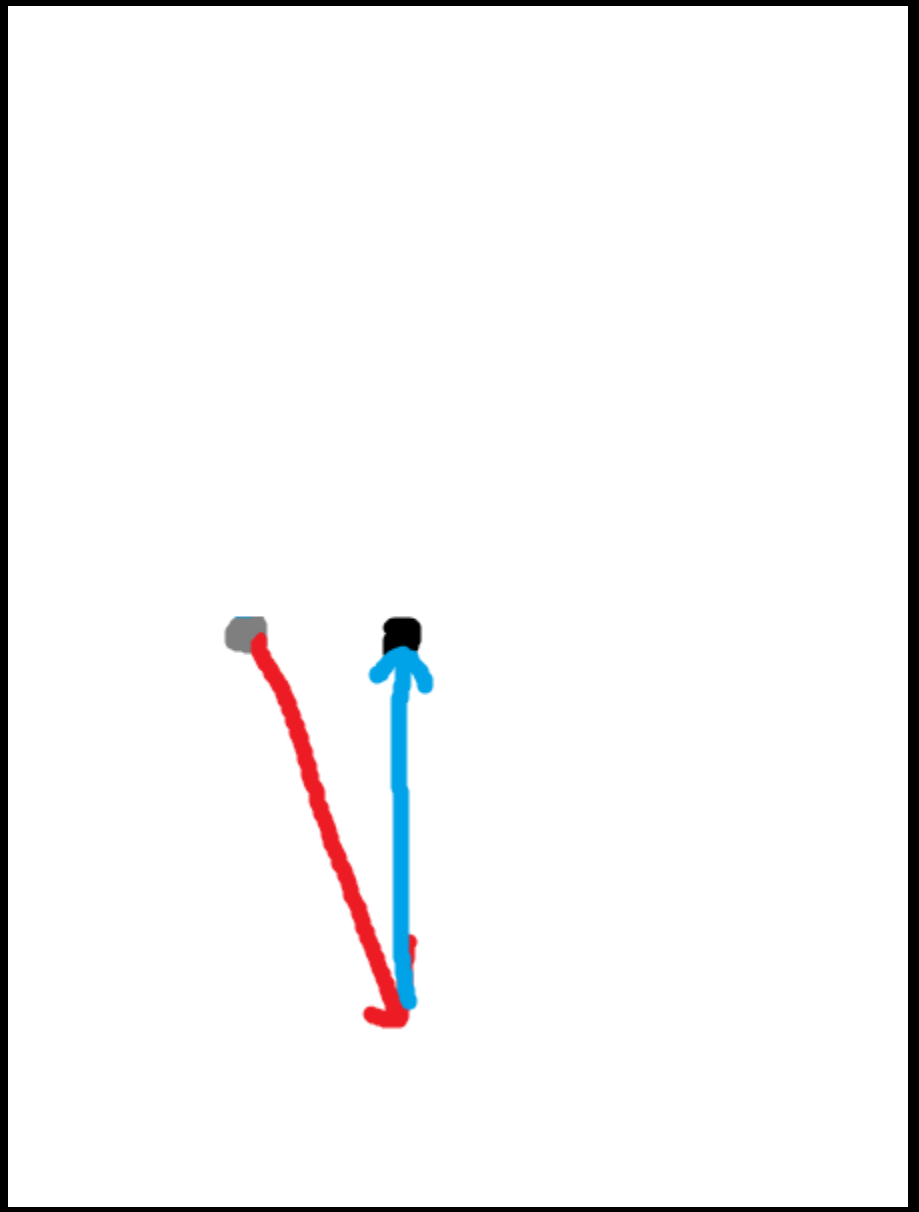
Case 4:

- A_0 = [0,5,0] -> [0,0,5,0,0,0]
- A_1 = [1,-5,0]
- B = [1,0,0]
- $A_0x + A_1y = B$
- x=1, y=1



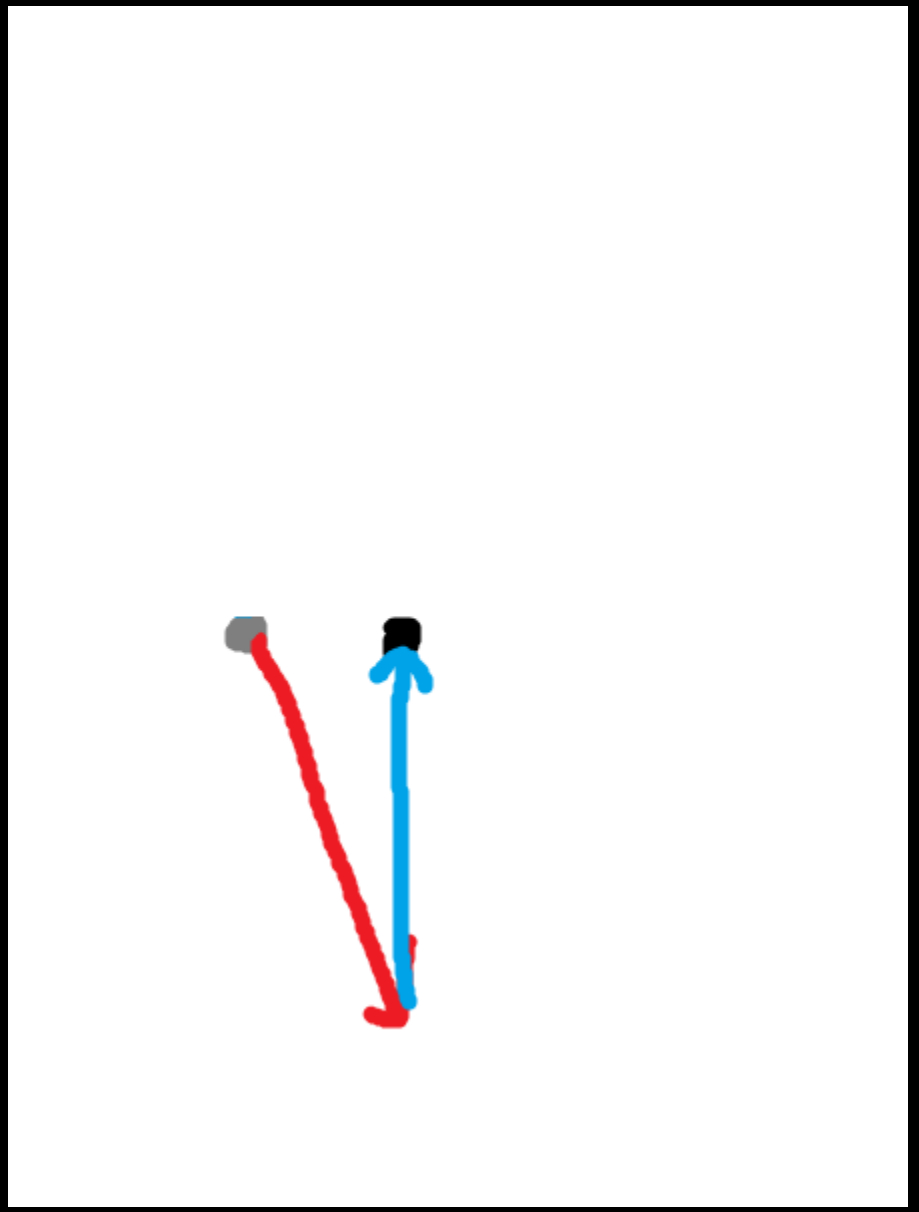
Case 4:

- A_0 = [0,5,0] -> [0,0,5,0,0,0]
- A_1 = [1,-5,0] -> [1,0,0,5,0,0]
- B = [1,0,0]
- $A_0x + A_1y = B$
- x=1, y=1



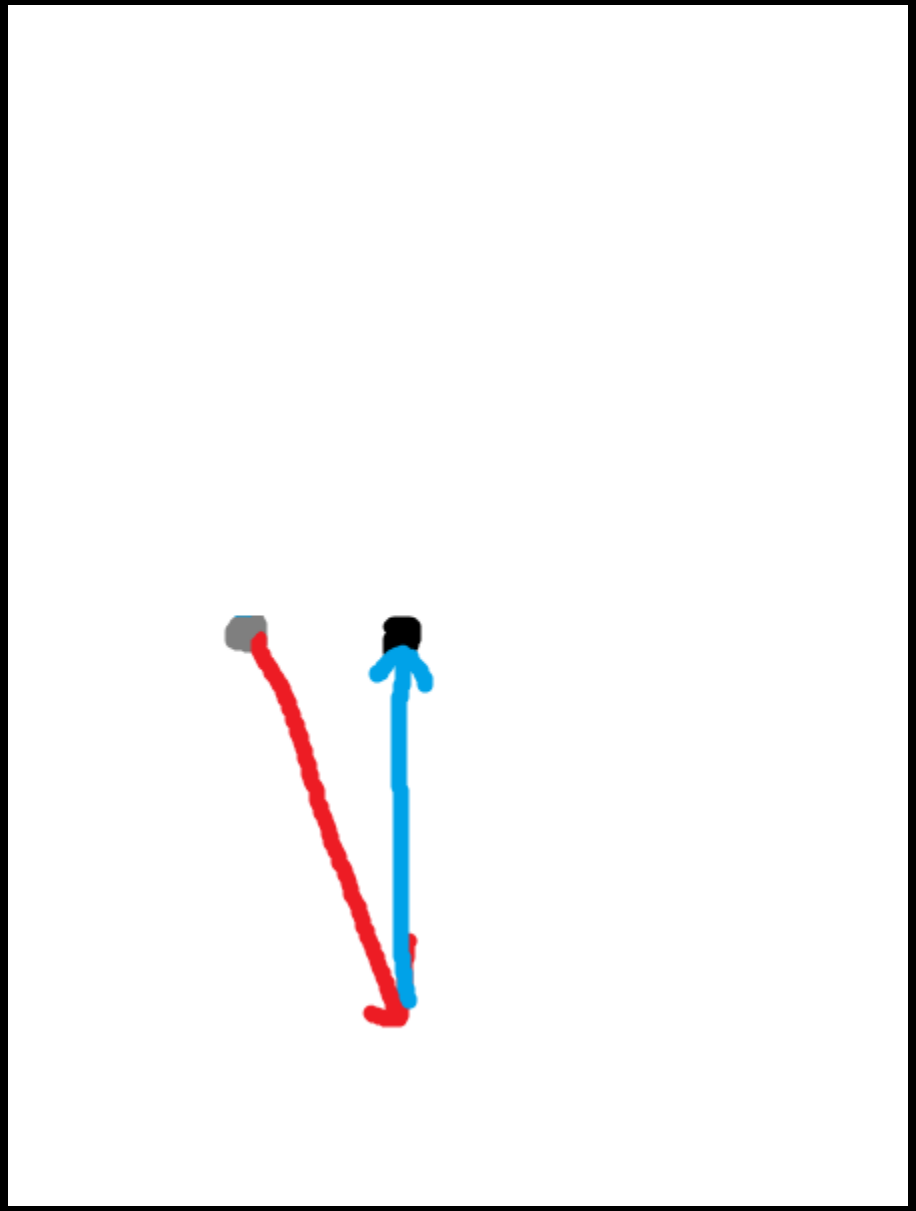
Case 4:

- A_0 = [0,5,0] -> [0,0,5,0,0,0]
- A_1 = [1,-5,0] -> [1,0,0,5,0,0]
- B = [1,0,0] -> [1,0,0,0,0,0]
- $A_0x + A_1y = B$
- x=1, y=1



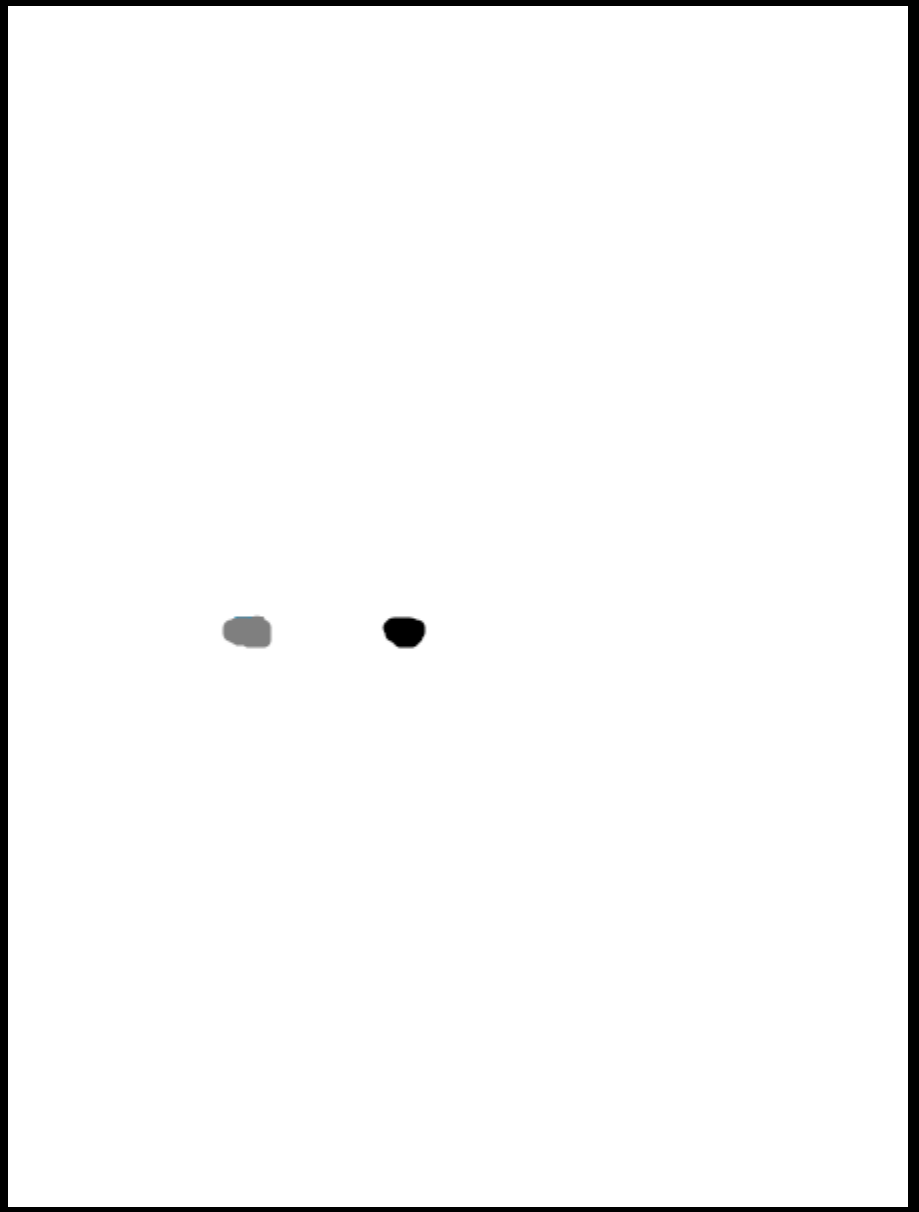
Case 4:

- A_0 = [0,5,0] -> [0,0,5,0,0,0]
- A_1 = [1,-5,0] -> [1,0,0,5,0,0]
- B = [1,0,0] -> [1,0,0,0,0,0]
- $A_0x + A_1y = B$
- x=1, y=1 -> [1,0,5,5,0,0]



Case 4:

- A_0 = [0,5,0] -> [0,0,5,0,0,0]
- A_1 = [1,-5,0] -> [1,0,0,5,0,0]
- B = [1,0,0] -> [1,0,0,0,0,0]
- $A_0x + A_1y = B$
- x=1, y=1 -> [1,0,5,5,0,0]
- x=0, y=0 -> [0,0,0,0,0,0]



System setup:

- Each shape has 150 joints
  - 6 channels per joint [+x,-x,+y,-y,+z,-z]
  - 900 channel vector
- Matrix is 900 rows with 70 columns
- Solve for 70 weights

Other pieces:

- Results are sparse
- Ignore blendshapes for eyelids and lips
  - Just use joint animation
- Do this for each region separately (8 times)

Part 4: Rendering

Quick Recap

- After all that work, what we have is:
  - Triangle Mesh
  - Diffuse Map
  - Normal Map
  - AO Map
- How to render?

Quick Overview:

- Skin SSS
- AO with spherical harmonics
- Adaptive Tessellation
- Eyes
- Teeth

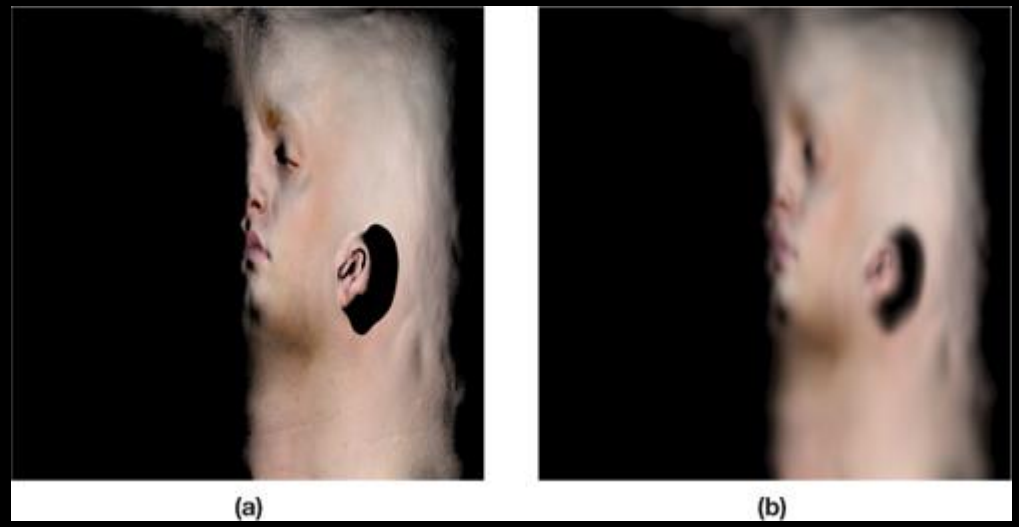
Skin SSS

- Remember the NVIDIA Doug Jones Head?
- http://http.developer.nvidia.com/GPUGems3/gpugems3_ch14.html
- “Bible” of skin shading
- Goal is to make it cheaper while retaining quality.

Skin Shader: Texture Space Blur

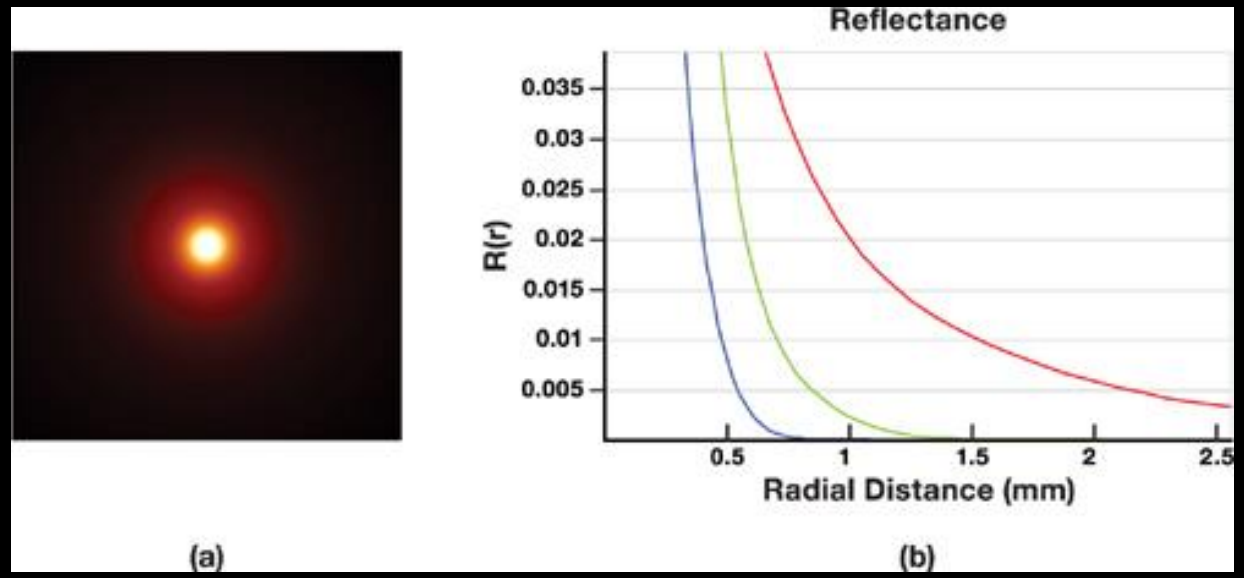
- How to actually do the blur?
- In theory, 5 gaussians
- Separable, but still many passes

	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0



Skin Shader: Texture Space Blur

- Discussion with Morten Mikkelson
- We both agreed: It’s a blur and a spike
- Exact shape of blur doesn’t matter
- Same conclusion in Ryse talk two days ago by Nicolas Schulz.

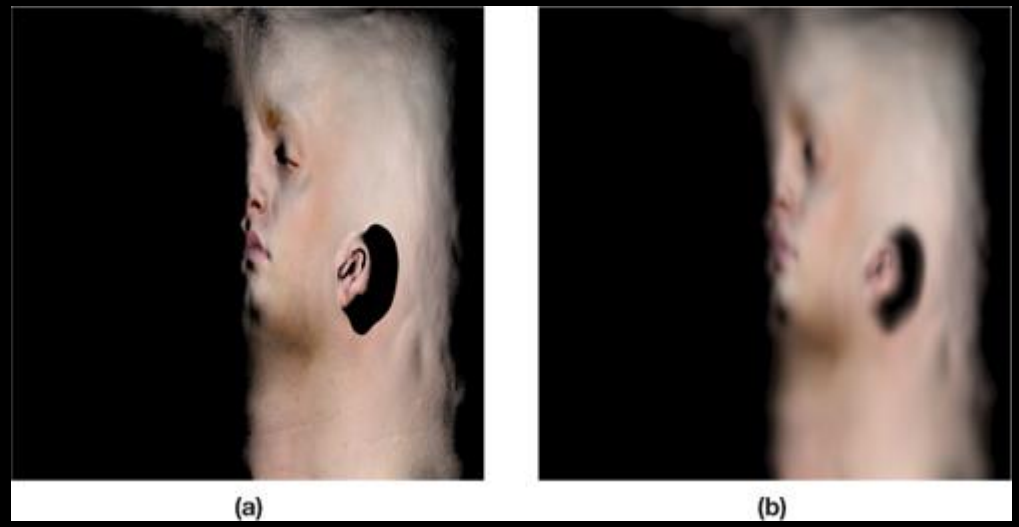


Skin Shader: Texture Space Blur

1. Render to lightmap
2. Blur vertically
3. Blur horizontally

- Combine all 5 blurs into one
- It’s wrong. Yet, I’m still able to sleep at night.
- How big is the map?

	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0



How Big? Rendering at 1080p.

Answer: 256x256

Answer: 256x256

Texture Space

- Less publicized advantage of texture space.
- Much lower texel density than the screen space size.
  - Because it’s a BLUR! It doesn’t need high frequency information.
- Texture space better in some cases, Screen Space better in others.

Skin Shader: One pass approximation.

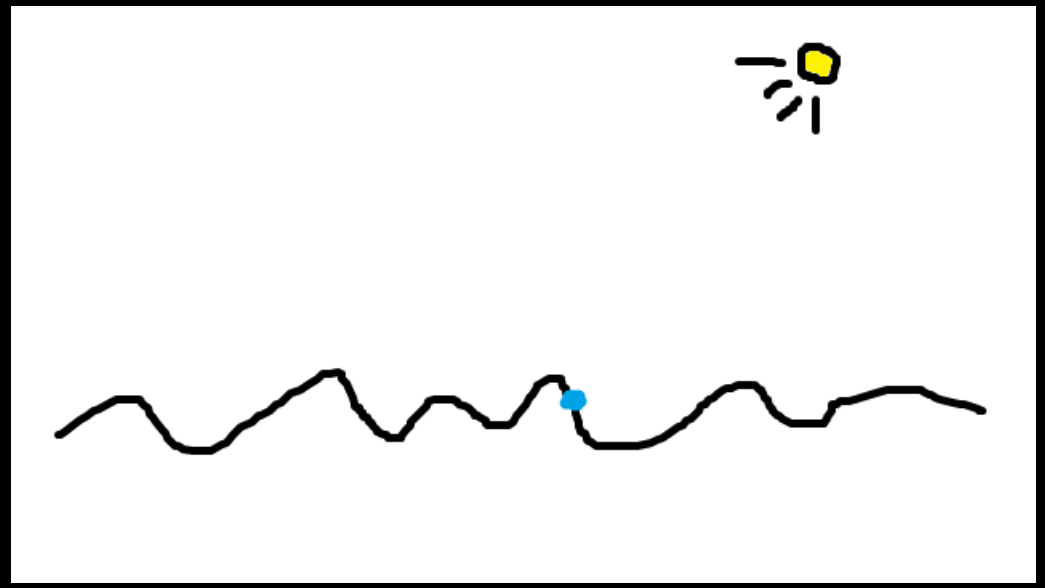
- We can also do it in one pass.

	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0

Skin Shader: One pass approximation.

- Top row with detail normal

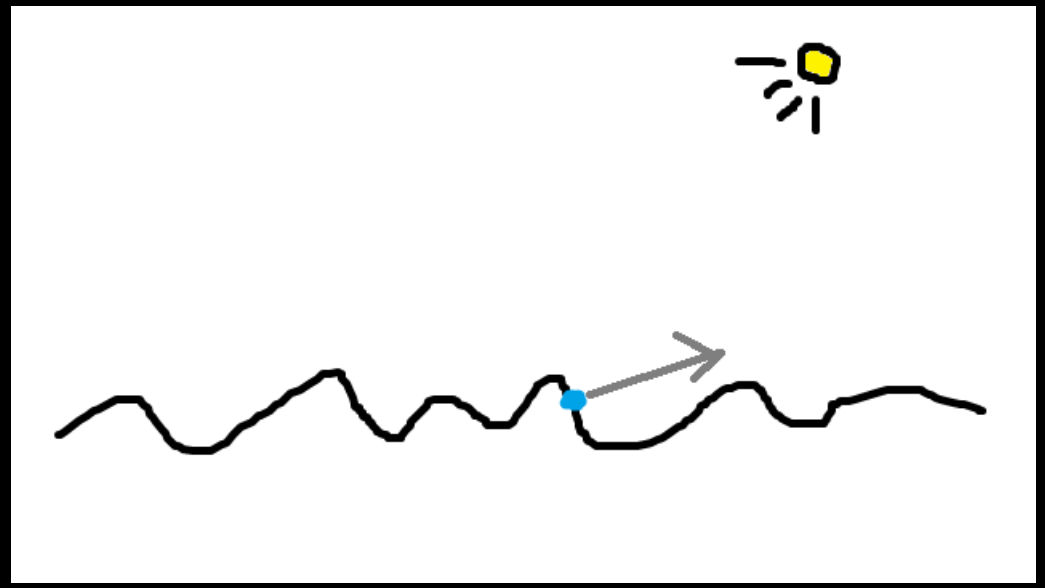
	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0



Skin Shader: One pass approximation.

- Top row with detail normal

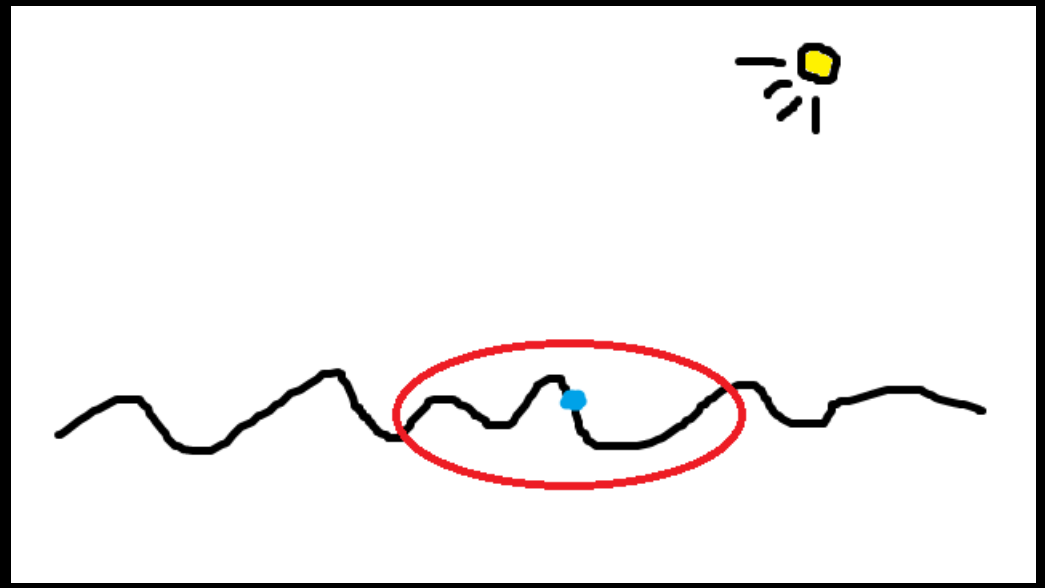
	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0



Skin Shader: One pass approximation.

- Blur accumulates nearby pixels

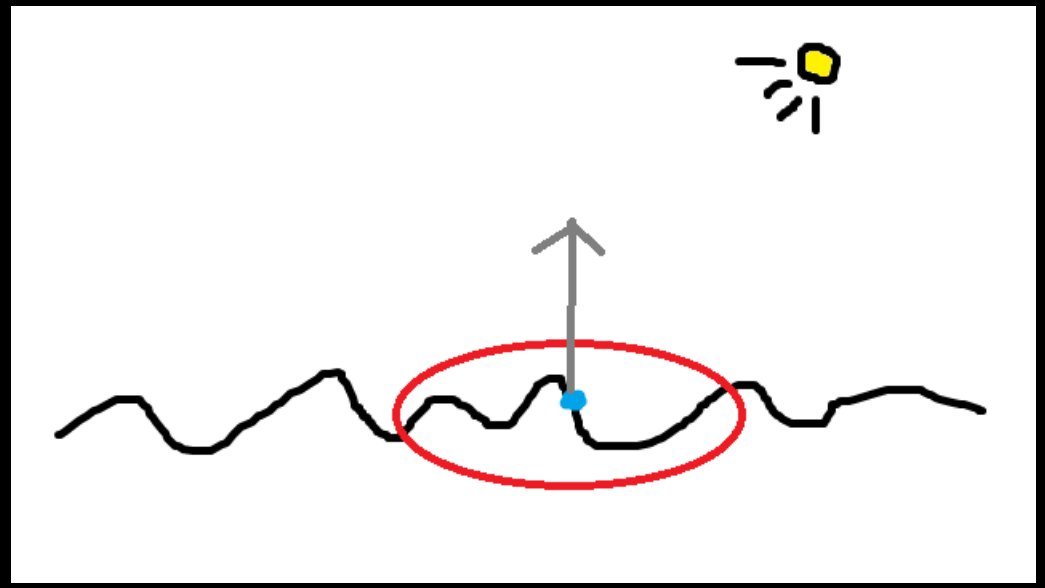
	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0



Skin Shader: One pass approximation.

- Assume that normals cancel out
- Approximate all samples with geometry normal

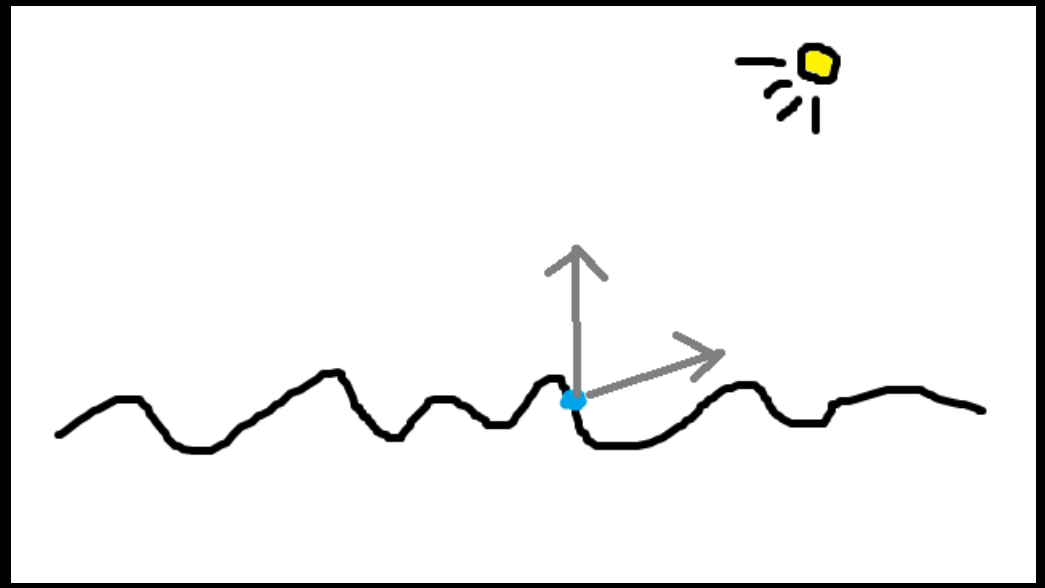
	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0



Skin Shader: One pass approximation.

- Top row lighting with detail normal
- Everything else with geometry normal

	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0



Skin Shader: One pass approximation.

- N = normal mapped normal
- G = geometry normal
- DiffN = DiffuseLight(N)
- DiffG = DiffuseLight(G)
- ColorRatio = (.233,.455,.649)
- Diff = lerp(DiffN,DiffG,ColorRatio)

	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0

Comparison

- Texture Blur

Comparison

- Geometry/Detail Blend

Comparison

- Lambert

Comparison

- Texture Blur

Comparison

- Geometry/Detail Blend

Comparison

- Lambert

Comparison

- Texture Blur

Comparison

- Geometry/Detail Blend

Comparison

- Lambert

Things that affect need for SSS

- Harsh lighting needs more SSS than flat lighting
- Farther characters don’t need SSS
  - SSS is only a few mm anyways
  - Less than one pixel far away

Powdered doughnuts:

Powdered doughnut problem:

- Evan Wells came up to me and said:

Powdered doughnut problem:

- Evan Wells came up to me and said (paraphrasing):

> “There’s a big problem with the cutscene! Drake looks like he ate a powdered doughnut!!”

Powder Doughnut

Solution: Shadow

Shadows

- Easy! Just add a shadow.
  - We have no limit on those, right?
- Better option?

Spherical Harmonic Ambient Occlusion

- It’s like ambient occlusion
- But with spherical harmonics!
- Bake SH per-vertex
- Figures out where light can and can not come from.

SHAO: No shadow or SHAO

SHAO: SHAO

SHAO: With shadow

SHAO: No shadow or SHAO

SHAO: SHAO

SHAO: With shadow

SHAO

- Full rendering
- 4 lights, only one with a shadow

SHAO: No shadow or SHAO

SHAO: SHAO

SHAO: With shadow

SHAO: With shadow and SHAO

Powder doughnut:

- Unshadowed lights causing highlights
- Can also see it in the nostrils, and ears
- Made worse by proper physically based shading
  - Bright rimlights

SHAO

- Not as good as a shadow.
- Way better than nothing.
- Too expensive to use everywhere (vertex cost)
  - A major help on unshadowed lights affecting the head.
  - Easy to use with forward rendering
  - Probably too hard deferred

Adaptive Tessellation

- We can have tessellation this generation.
- Yes, we say that every generation.
- But now I mean it!
- Tessellating the whole thing is too expensive in all but extreme cases.
  - But can we tessellate only the silhouette

Tessellation Off

Tessellation Adaptive

Tessellation Everywhere

Tessellation Off

Tessellation Adaptive

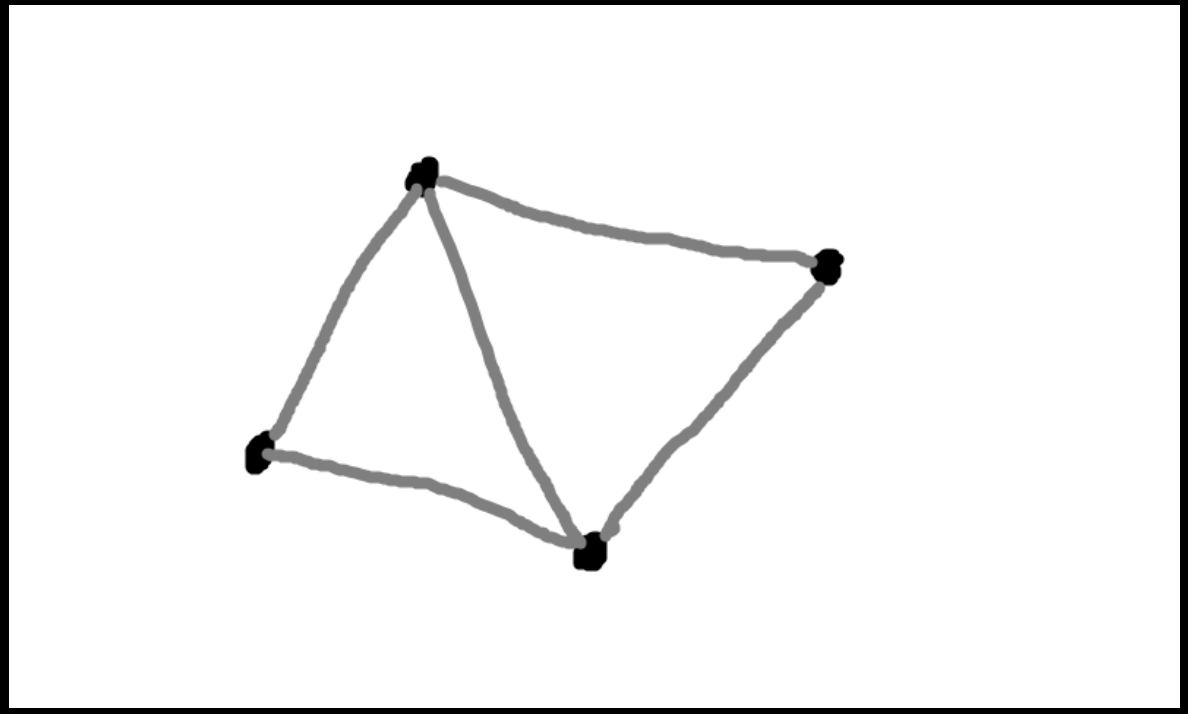
Tessellation Everywhere

Tessellation Adaptive

- Times on NVIDIA 660ti
- Off: 4.14ms
- Adaptive: 4.22ms
- Everywhere: 6.18ms

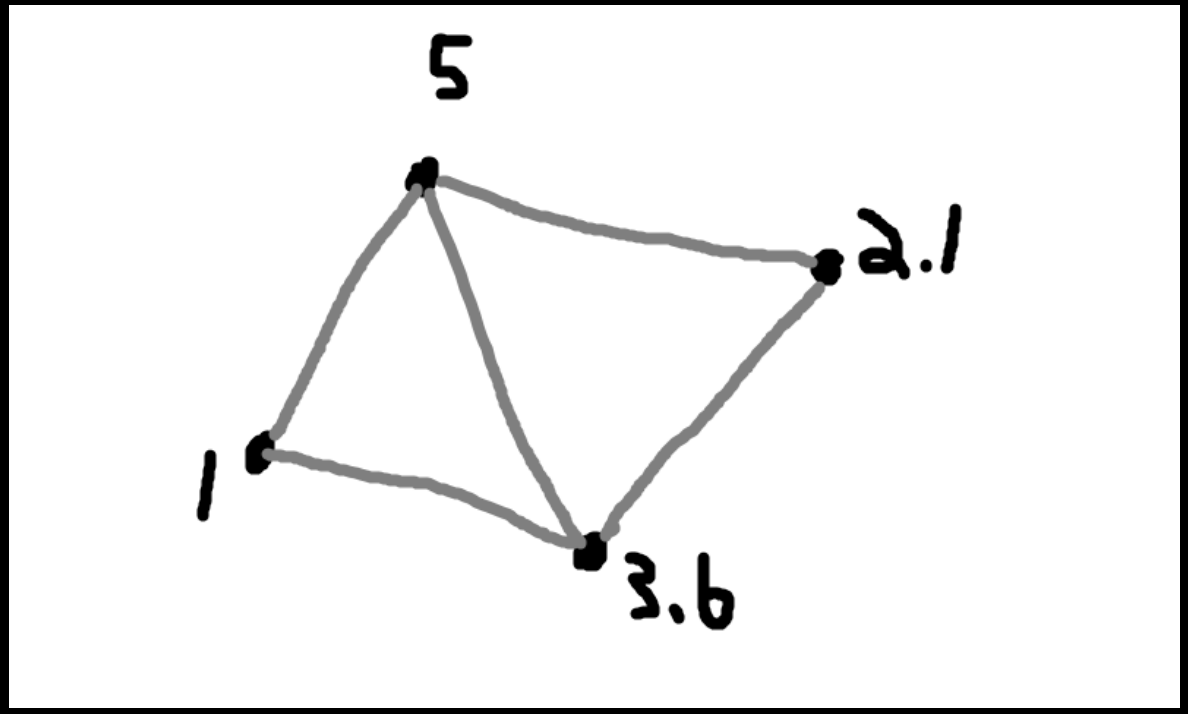
Adaptive Tessellation

- DX11 is awesome
- Hull shader with fractional_odd partitioning
- Each vertex calculates “tessellation weight”
- Edge tessellation commutative function of both verts



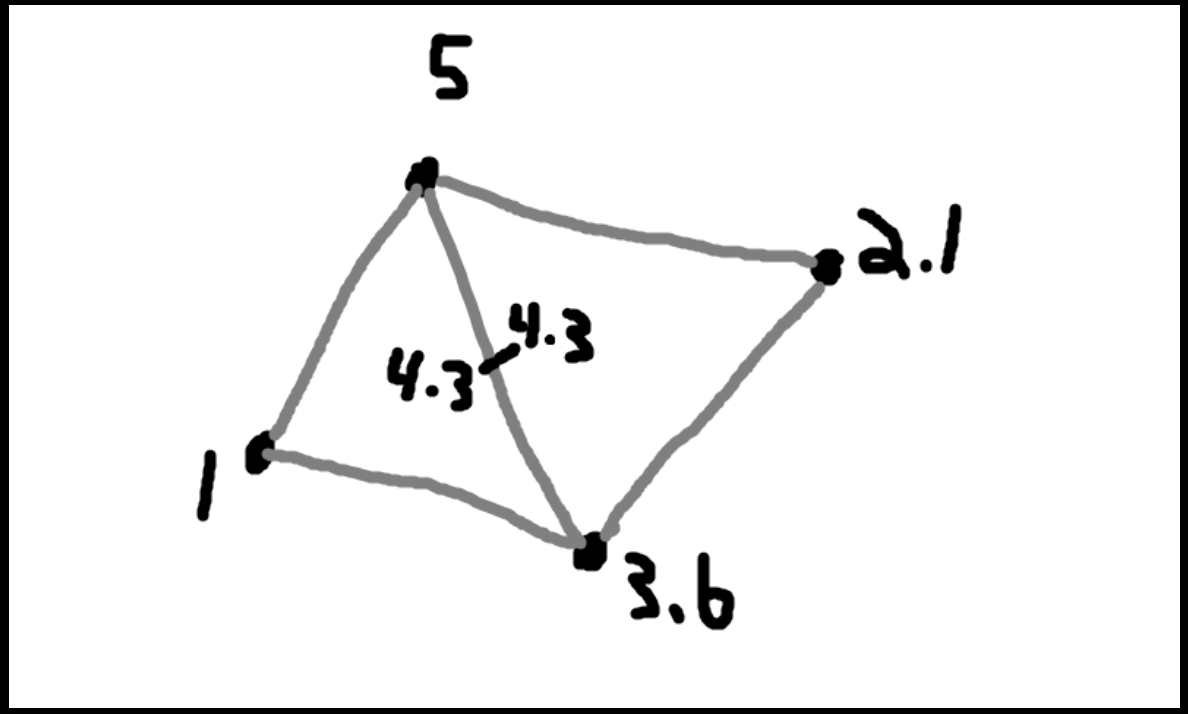
Adaptive Tessellation

- DX11 is awesome
- Hull shader with fractional_odd partitioning
- Each vertex calculates “tessellation weight”
- Edge tessellation commutative function of both verts



Adaptive Tessellation

- DX11 is awesome
- Hull shader with fractional_odd partitioning
- Each vertex calculates “tessellation weight”
- Edge tessellation commutative function of both verts



Adaptive Tessellation

- Const hull shader

```
HS_CONSTANT_DATA_TRI PhongTessConstHS( InputPatch<VS_OUTPUT_HEAD, 3> input )
{
    HS_CONSTANT_DATA_TRI output;

    float tess0 = input[0].TessVal;
    float tess1 = input[1].TessVal;
    float tess2 = input[2].TessVal;

    output.Weights = float3(tess0,tess1,tess2);

    float contrast = 1.2;
    output.MaxVal = saturate(contrast*((tess0 + tess1 + tess2)/3.0f));

    output.Edges[0] = max(0,.5*(tess1+tess2)*g_tessFactorEdge)+1.0;
    output.Edges[1] = max(0,.5*(tess2+tess0)*g_tessFactorEdge)+1.0;
    output.Edges[2] = max(0,.5*(tess0+tess1)*g_tessFactorEdge)+1.0;
    output.Interior[0] = output.MaxVal*g_tessFactorEdge;

    return output;
}
```

Adaptive Tessellation

- Hull shader

```
[domain("tri")]
[partitioning("fractional_odd")]
[outputtopology("triangle_cw")]
[outputcontrolpoints(3)]
[patchconstantfunc("PhongTessConstHS")]

VS_OUTPUT_HEAD PhongTessHS( InputPatch<VS_OUTPUT_HEAD,3> patch,
                            uint index : SV_OutputControlPointID)
{
    return patch[index];
}
```

Adaptive Tessellation

- Domain shader
- Phong Tesselation

```
[domain("tri")]
VS_OUTPUT_HEAD PhongTessDS(HS_CONSTANT_DATA_TRI input,
                           float3 uv : SV_DomainLocation,
                           const OutputPatch<VS_OUTPUT_HEAD, 3> patch)
{
    VS_OUTPUT_HEAD ret = (VS_OUTPUT_HEAD)0;

    AccumulatVertex(ret,uv.x,patch[0]);
    AccumulatVertex(ret,uv.y,patch[1]);
    AccumulatVertex(ret,uv.z,patch[2]);

    float3 wpBase = ret.WorldPos;
    float3 normBase = ret.Normal;

    float3 wp0 = ProjectPoint(patch[0].WorldPos,patch[0].Normal,wpBase);
    float3 wp1 = ProjectPoint(patch[1].WorldPos,patch[1].Normal,wpBase);
    float3 wp2 = ProjectPoint(patch[2].WorldPos,patch[2].Normal,wpBase);

    float len0 = DistLinePlane(patch[0].WorldPos,patch[0].Normal,wpBase,normBase);
    float len1 = DistLinePlane(patch[1].WorldPos,patch[1].Normal,wpBase,normBase);
    float len2 = DistLinePlane(patch[2].WorldPos,patch[2].Normal,wpBase,normBase);

    float alpha = input.Weights.x*uv.x + input.Weights.y*uv.y + input.Weights.z*uv.z;
    float tessFactorAlpha = alpha * g_tessFactorAlpha;

    float3 wpDst = tessFactorAlpha*(uv.x*wp0 + uv.y*wp1 + uv.z*wp2) + (1.0f-tessFactorAlpha)*wpBase;

    float4 projPos = mul( g_mWorldViewProj, float4(wpDst,1.0) );

    ret.Position = projPos;

    return ret;
}
```

Eyes: Shader

Eyes: Big thing

- AO

Eyes: Big thing

- AO baked into Lat-Long space.

Eyes: Shader

Eyes: Shader

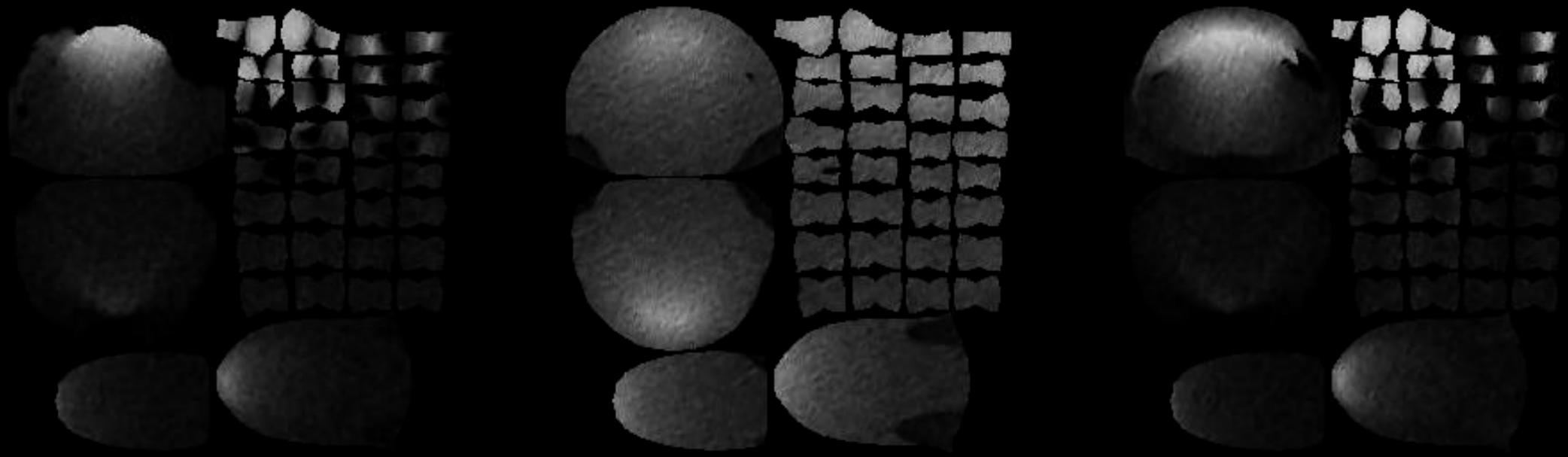
- Compress all 70 maps with PCA
- Drive using Y of eye joints

Teeth: Big thing

- AO

Teeth: Big thing

- AO based on light source in front



Teeth: AO

Teeth: AO

Teeth: AO

Teeth: AO

Teeth: AO

- Compress all 70 maps with PCA
- Drive with Y of mouth joints

Conclusions:

- Overall, it works
- Diffuse animation is essential
- Animation doesn’t always use the best blendshapes
  - Wrinkles under the eyes
  - Would be better solved to a proper rig
- Real games need ways to adjust the meshes
  - Art director wants a smaller nose, don’t want to resculpt 70 expressions

Credits:

- Infinite-Realities (3d scans)
- Ramahan Faulk (character modeling)
- Mocap Militia (mocap shoot)
- Matthew Mercer (mocap talent)
- Special Thanks: HP Duiker, Maggie Bellomy, Habib Zargarpour

Questions?