GDC

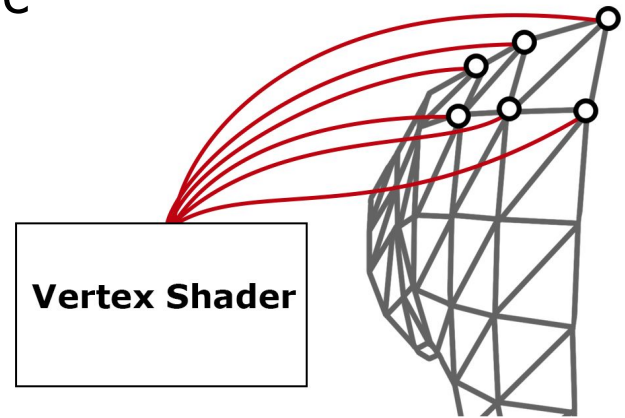
The Illusion of Motion

Making magic with textures in the vertex shader

Mario Palmero
Lead Programmer at Tequila Works

Dark Ages before
Textures in the Vertex Shader

- What is the Vertex Shader?
  - A programmable part of the graphic pipeline
  - Vertex properties can be modified
    - Position
    - Normal
    - Vertex color
    - UVs



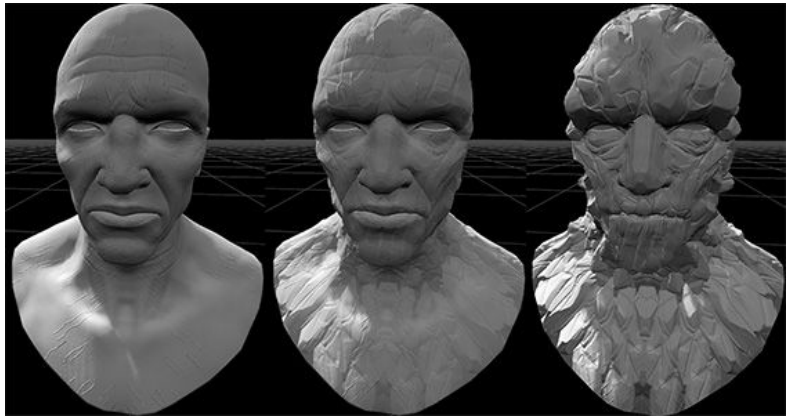
Dark Ages before
Textures in the Vertex Shader

- No textures in vertex shader :(
- Use vert attributes instead, such as vertex color and UVs.
  - Vert Positions
  - Normals
  - Pivot Points

Textures in Vertex Shader

Since DirectX 11 we can cheaply sample textures in the vertex shader.

- Typical use case: Displacement Map



Example from UE4 documentation

Textures in Vertex Shader

However there are more interesting use cases!

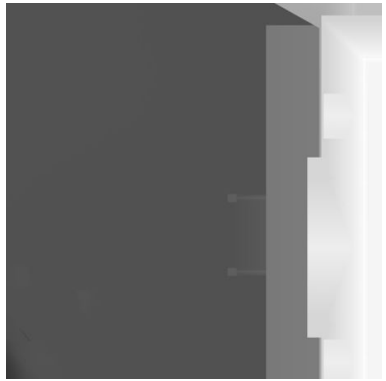
- Pixels are cells of information
- Vectors of 3 or 4 elements
- Different levels of precision

The data in a texture can be used for different things. **Be creative!**

Some Simple but Creative Uses

Where does the rain stop?

- Read the height from map
- Reset Z animation



Some Simple but Creative Uses

Fixed-to-the-ground meshes

- Read the height from map
- Add as an offset to the mesh
- Easy to place for artists

Some Simple but Creative Uses

Fixed-to-the-ground meshes

Before

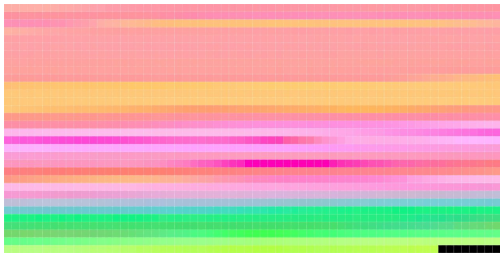
Now

Some Simple but Creative Uses

Smoke animation in a texture

- Baking XYZ of the hand as RGB
- Each frame is a pixel
- Uncompressed texture
- Position normalized: 0 to 1
- Position scaled back in the shader

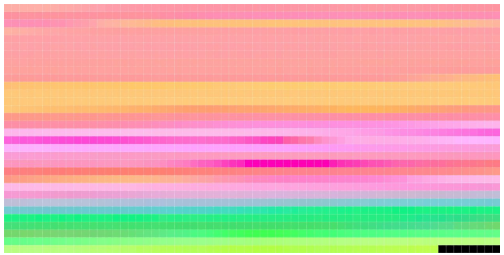
Collaboration with Simon Trümpler



Some Simple but Creative Uses

Smoke animation in a texture

- Smoke has several rings of height
- Bottom ring reads current pixel/frame
- Previous pixels for rings above



Pre-calculated Particle Animation

- 4001 particles
- 180 frames of animation

Pre-calculated Particle Animation

Positions as pixels

- Each pixel is a position
- Each row is a frame
- Each column is the position of a vertex along time

Pre-calculated Particle Animation

- Texture Dimension: 4001x180
- Texture Size: 2813 KB
- Texture Format: B8G8R8A8
- Bilineal filtering of texture used to interpolate between positions

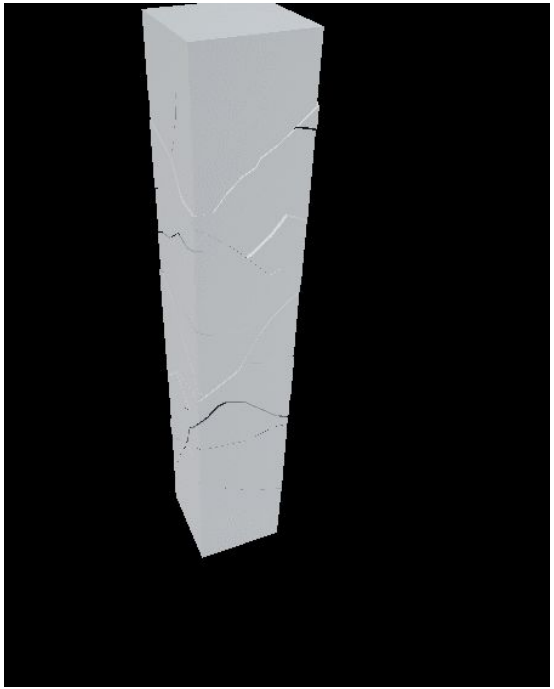
Optimization: Reduce it vertically (half) by deleting every other frame.

Pre-calculated Particle Animation

- Uncompressed texture
- XYZ as RGB
- Normalized values to fit in the 0 to 1 range
- Positions are scaled back in the shader
- Precision is obviously lost

For additional precision a 16 bit texture can be used instead.

Pre-calculated Rigid Object Animation

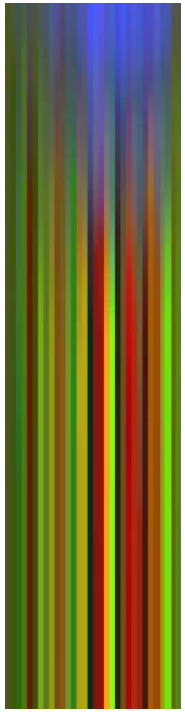


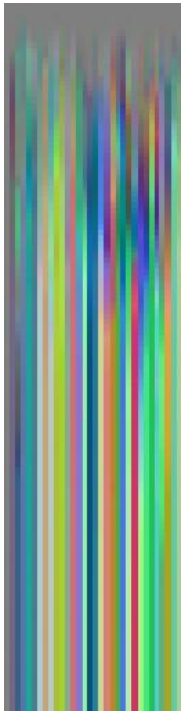
Pre-calculated Rigid Object Animation

Pivoting

- Groups of vertices sharing a pivot
- Every vertex in a group shares movement
- Memory improvement over brute force

Pre-calculated Rigid Object Animation





- 32 objects
- 128 frames of animation
- 1 Texture for Position, 1 for Rotation
- Texture Dimension: 32x128
- Texture Size: 16KB Pos + 16KB Rot
- Texture Format: B8G8R8A8
- Quaternion as RGBA

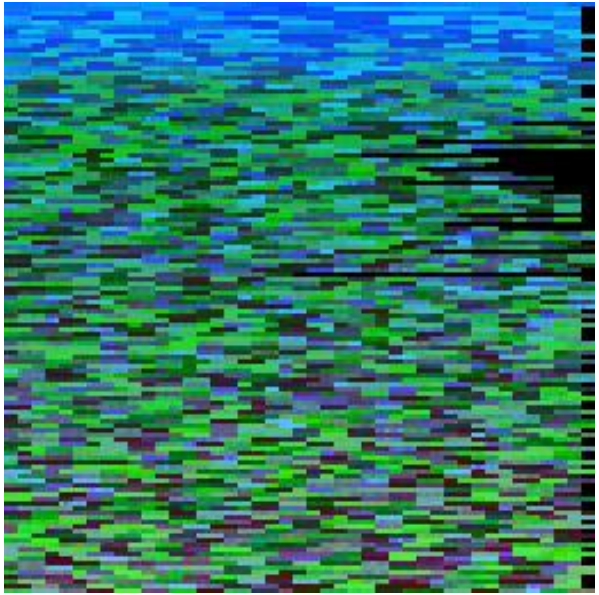
Optimization: Reduce it vertically (half) by deleting every other frame.

Vertex-count-agnostic
Morph Targets

Vertex-count-agnostic
Morph Targets

- The position texture

- Each row of pixels is a morph target



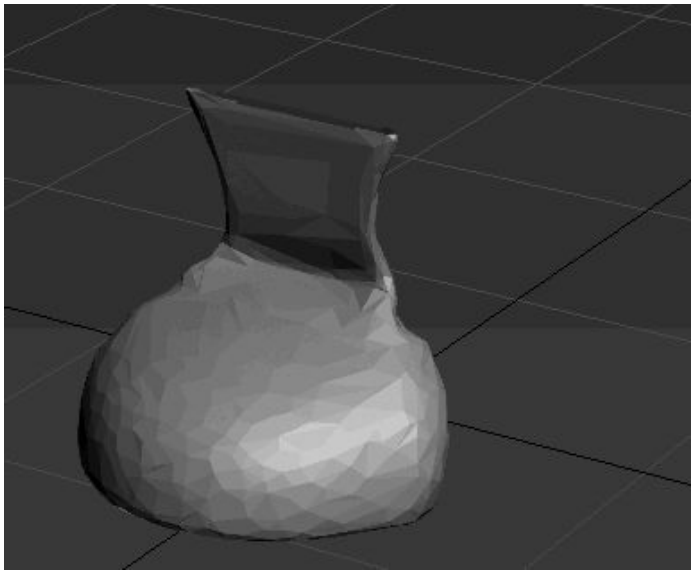
This is a cropped portion of the vert position texture

Vertex-count-agnostic
Morph Targets

- Every pixel represents a vert position. Three pixels represent one triangle.
- If three pixels are black it means that that triangle is not needed for that frame and so its vert positions will be 0,0,0. The triangle will virtually cease to exist.

Vertex-count-agnostic Morph Targets

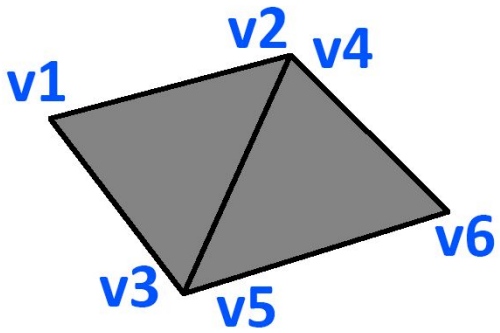
- The Morphing

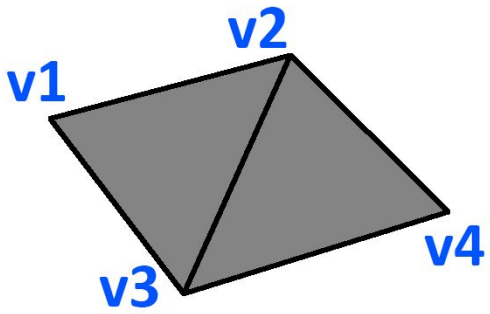


- In the shader we don't linearly interpolate, instead we jump to the next morph

Vertex-count-agnostic
Morph Targets

- Optimization: Triangle Pairing
  - We can reduce the memory footprint by pairing as many triangles as possible. We can reduce up to 33%



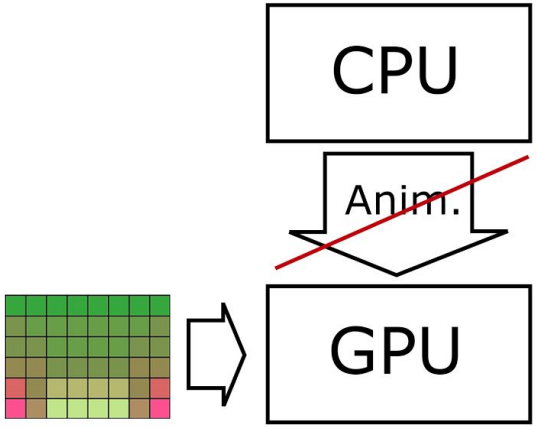


Vertex-count-agnostic
Morph Targets

- Optimization: But ideally...
  - We should have different options of geometry compression

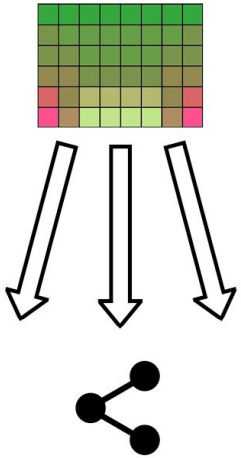
Advantages of
Textures as data containers

Access



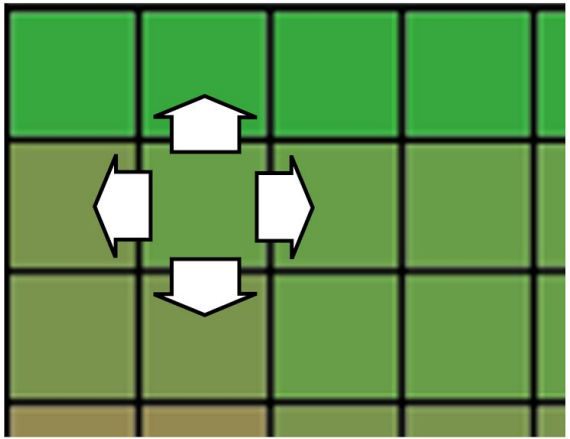
No CPU involved

Shareable



Standard format

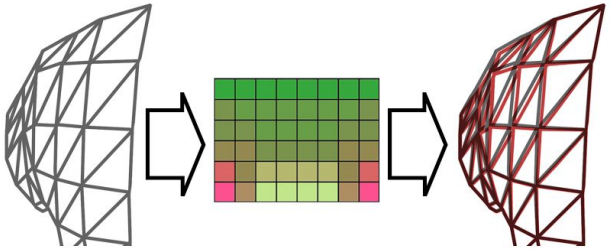
Order



Intrinsic spatial coherence

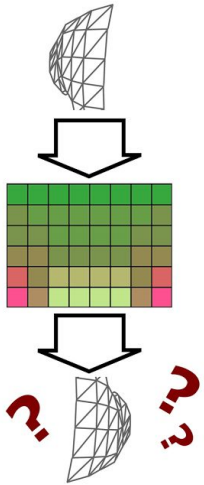
Drawbacks of
Textures as data containers

Precision



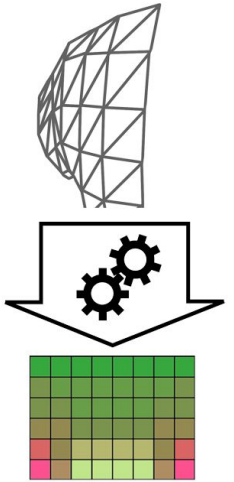
Limited precision format

Debugging



Difficult to follow the track

Creation



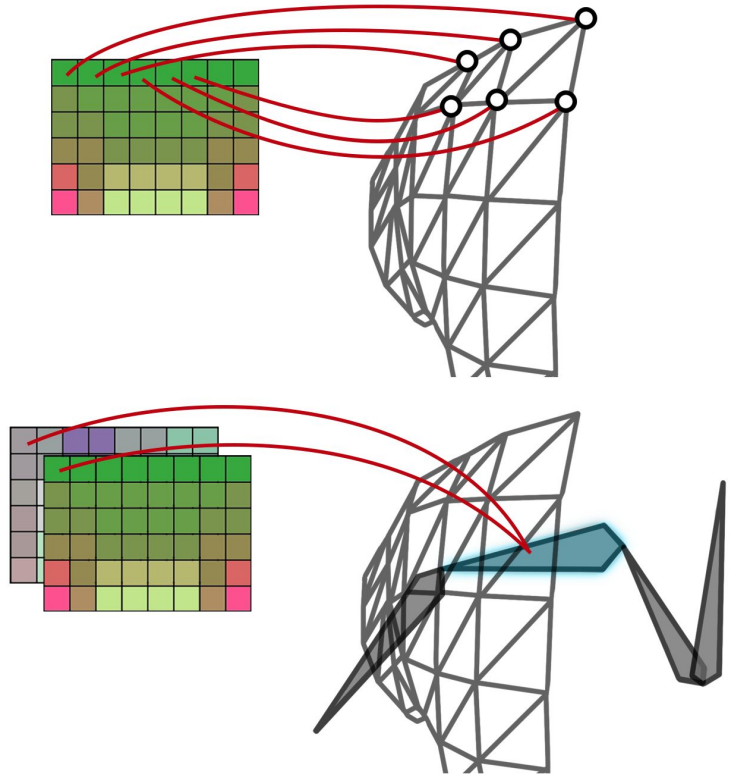
Custom tools

Thinking about
Baking animation and cloth simulation

- Huge amount of vertex data
- Compress the data depending on the amount of movement
- But during a pee-break I had an **idea**

The Idea!

- What if we transform simulations into bone animation?
- What if we put the bone transformation data into a texture?



Can We Actually Do It?

- The skinning of the mesh is already being done in the GPU
  - So theoretically, yes

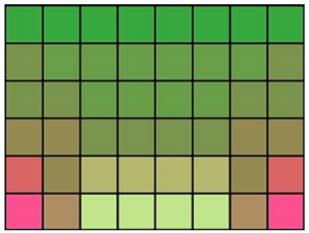
- But we need to have access to all the information the CPU provides to the GPU in the conventional way
  - Through mesh and texture information

To achieve that
Info needed

- Store the translation of bones

To achieve that
Info needed

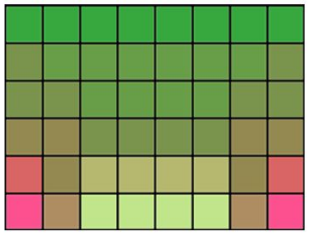
- Store the translation of bones → **A texture**



position

To achieve that
Info needed

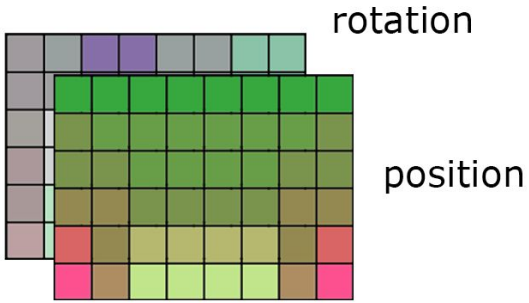
- Store the translation of bones → A texture
- **Store the rotation of bones**



position

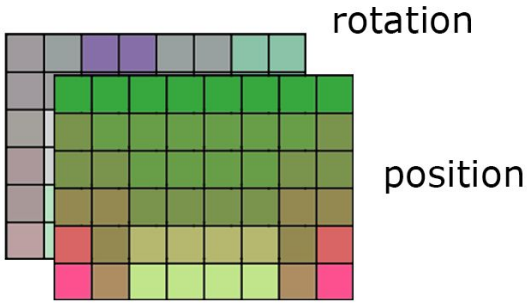
To achieve that
Info needed

- Store the translation of bones → A texture
- Store the rotation of bones → **Another texture**



To achieve that
Info needed

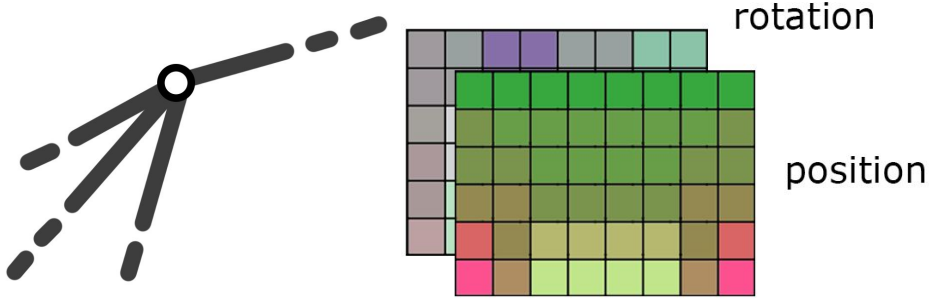
- Store the translation of bones → A texture
- Store the rotation of bones → Another texture
- **Store the weighting of the vertices**



To achieve that
Info needed

- Store the translation of bones → A texture
- Store the rotation of bones → Another texture
- Store the weighting of the vertices → **UV channels**

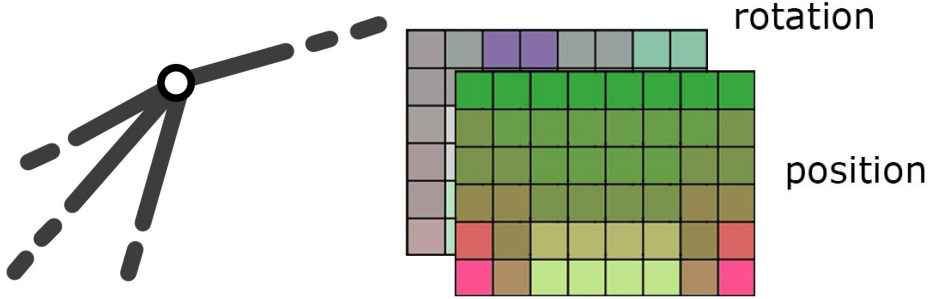
Second UV channel
[weightBone1, weightBone2]
Third UV channel
[weightBone3, weightBone4]



To achieve that
Info needed

- Store the translation of bones → A texture
- Store the rotation of bones → Another texture
- Store the weighting of the vertices → UV channels
- **Store the index of the relevant bones**

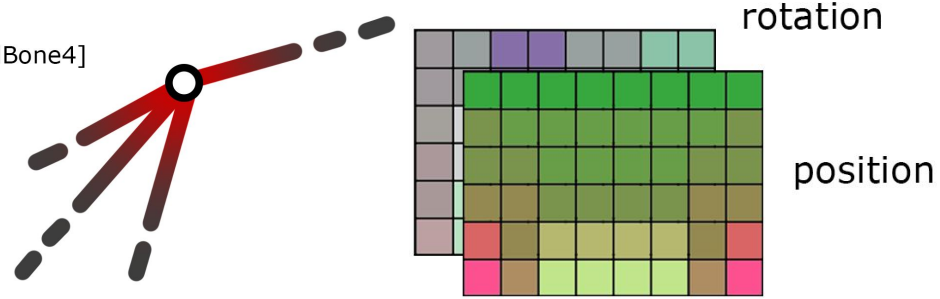
Second UV channel
[weightBone1, weightBone2]
Third UV channel
[weightBone3, weightBone4]



To achieve that
Info needed

- Store the translation of bones → A texture
- Store the rotation of bones → Another texture
- Store the weighting of the vertices → UV channels
- Store the index of the relevant bones → **Vertex color**

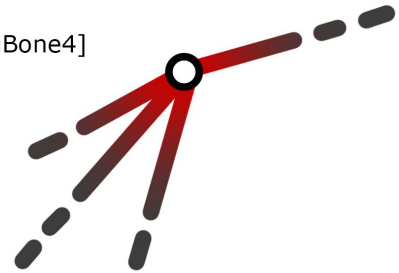
Vertex Color
[idBone1, idBone2, idBone3, idBone4]
Second UV channel
[weightBone1, weightBone2]
Third UV channel
[weightBone3, weightBone4]

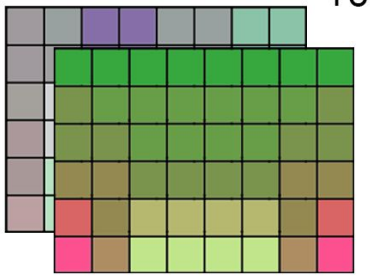


To achieve that
Info needed

- Store the translation of bones → A texture
- Store the rotation of bones → Another texture
- Store the weighting of the vertices → UV channels
- Store the index of the relevant bones → Vertex color
- **Store the initial offset of the bones**

Vertex Color
[idBone1, idBone2, idBone3, idBone4]
Second UV channel
[weightBone1, weightBone2]
Third UV channel
[weightBone3, weightBone4]

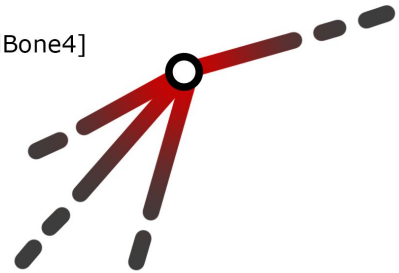


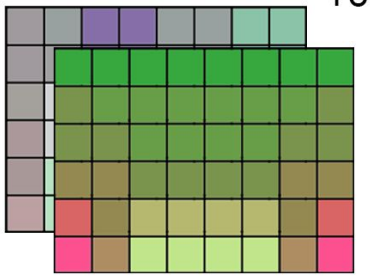


To achieve that
Info needed

- Store the translation of bones → A texture
- Store the rotation of bones → Another texture
- Store the weighting of the vertices → UV channels
- Store the index of the relevant bones → Vertex color
- Store the initial offset of the bones → **More textures!**

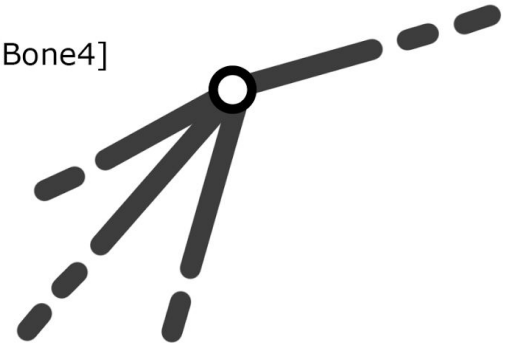
Vertex Color
[idBone1, idBone2, idBone3, idBone4]
Second UV channel
[weightBone1, weightBone2]
Third UV channel
[weightBone3, weightBone4]

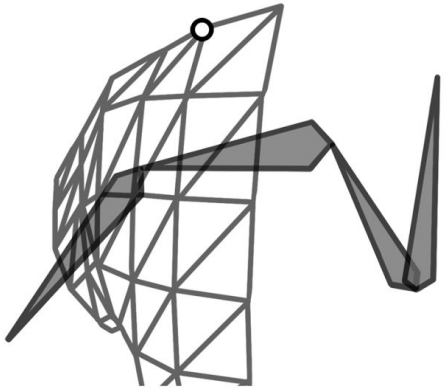




How We Actually Do It

Vertex Color
[idBone1, idBone2, idBone3, idBone4]
Second UV channel
[weightBone1, weightBone2]
Third UV channel
[weightBone3, weightBone4]

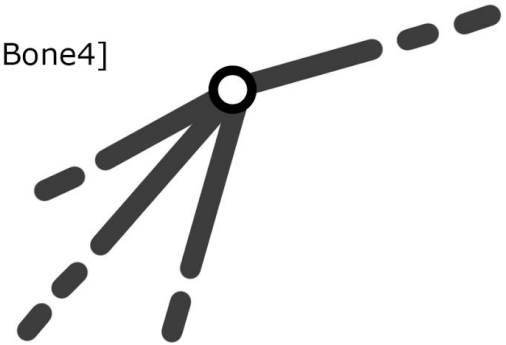


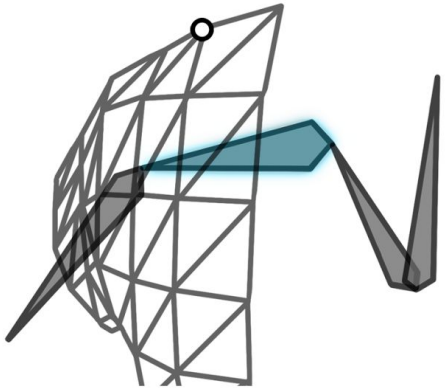


How We Actually Do It

- Read the indices of bones affecting the vertex from the vertex color (limitation of 256 bones)

Vertex Color
[**idBone1**, idBone2, idBone3, idBone4]
Second UV channel
[weightBone1, weightBone2]
Third UV channel
[weightBone3, weightBone4]

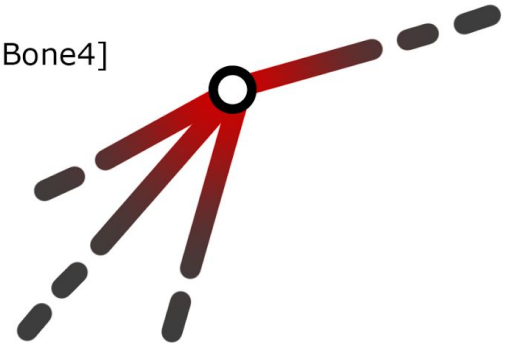


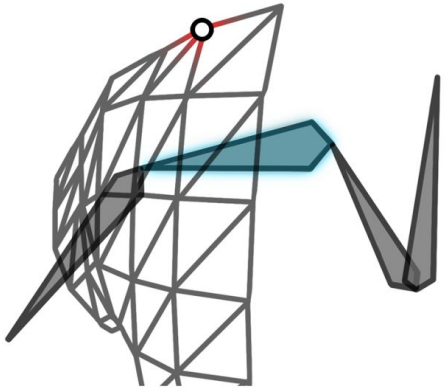


How We Actually Do It

- Read the influence of those bones over the vertex from the UV channels

Vertex Color
[**idBone1**, idBone2, idBone3, idBone4]
Second UV channel
[**weightBone1**, weightBone2]
Third UV channel
[weightBone3, weightBone4]



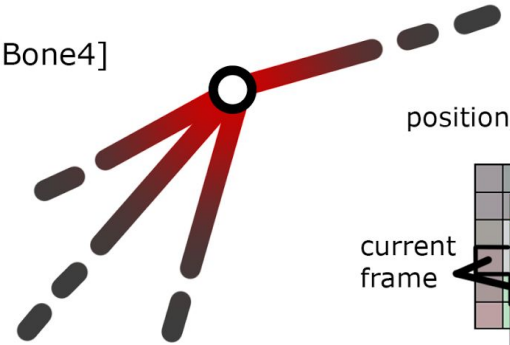


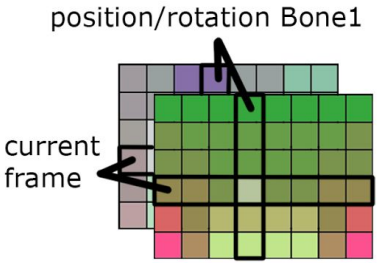
How We Actually Do It

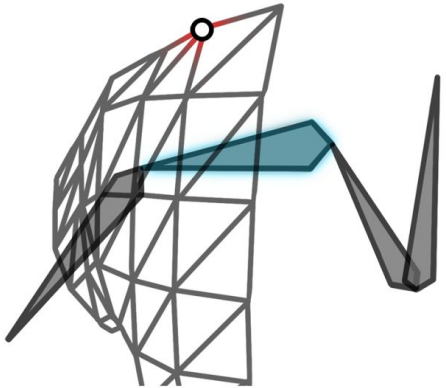
- Read the position and rotation of those bones from both textures

  We use the object position and the original bone offset from the texture to calculate final positions

Vertex Color
[**idBone1**, idBone2, idBone3, idBone4]
Second UV channel
[**weightBone1**, weightBone2]
Third UV channel
[weightBone3, weightBone4]



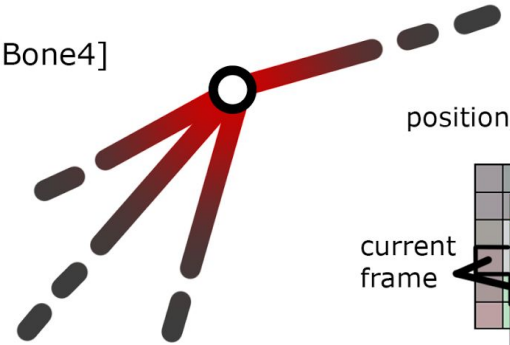


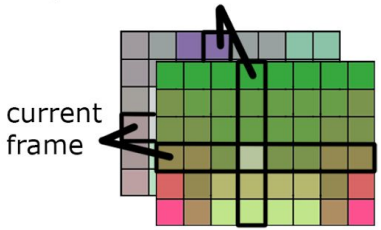


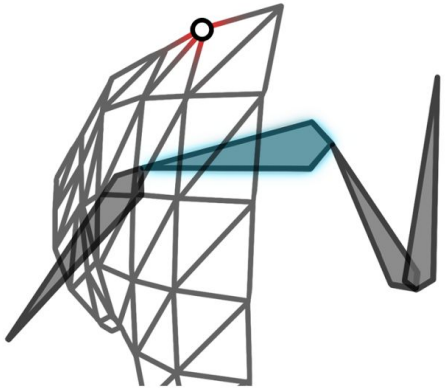
How We Actually Do It

- We have all what we need to do apply the linear skinning algorithm

Vertex Color
[**idBone1**, idBone2, idBone3, idBone4]
Second UV channel
[**weightBone1**, weightBone2]
Third UV channel
[weightBone3, weightBone4]





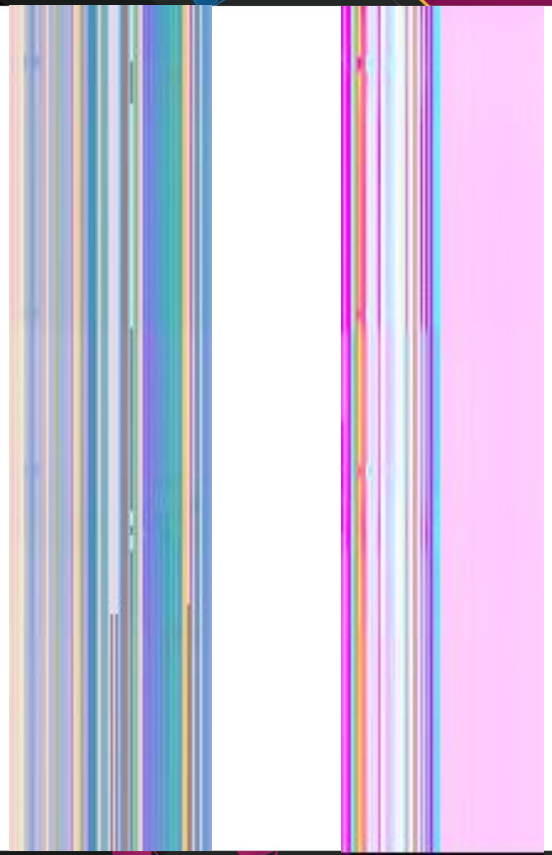


Final result

A full animation

Final result
Through another lens

We can take a look at
the textures that create
the previous animation



What Are
The Numbers?

- How much animation can we store in a texture?
- Can we handle facial animation for all the cinematics?
- Which frame rate are we talking about?

What Are
The Numbers?

166 minutes

- Of facial animation (56 bones)
- In two 4096x4096 textures (rotation and translation)
- 30 frames per second

What Are The Numbers?

21 minutes

- Of **awesome** facial animation (450 bones)
- In two 4096x4096 textures (rotation and translation)
- 30 frames per second

Future
Improvements

- Road map
  - Normal Solving
  - Animation Blending
  - Bone Scaling
  - Compression

Future Improvements

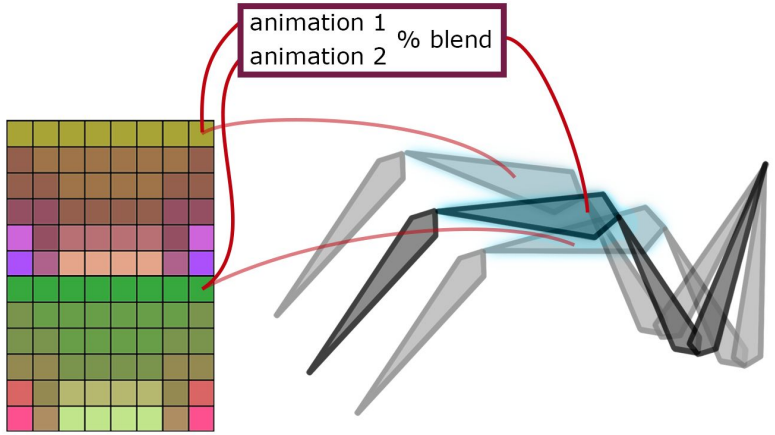
- Improve Normal Solving

  Rotating the normal by the quaternion of the bone is the simplest way to recalculate normals, but there are more accurate solutions

Future Improvements

- Implement Animation Blending

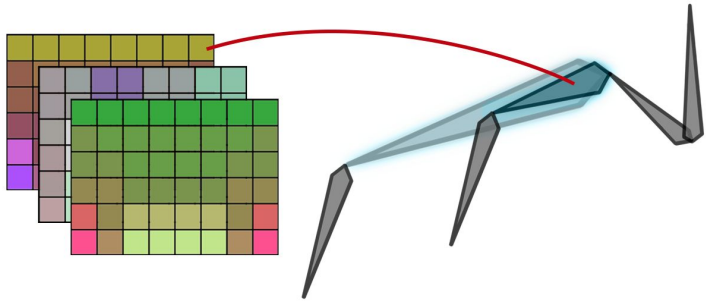
  The presented solution is useful for cinematic animations, but some improvements can be made to support animation blending.



Future
Improvements

- Support Bone Scaling

Transformation and rotation is already supported but we can add scaling with the use of another texture.



Future Improvements

- Enhance Compression

  We are storing information for each frame (30 fps), but that can be improved with a smarter compression and interpolation system.

Final Thoughts

- Know your tools
- Textures can be used as data containers
- Approach problems from different perspectives

Annex

This technique uses bone information. What about blendshapes or simulations?

- This issue is already solved by other people that also inspired our work: https://vimeo.com/123883474

Thanks!

- Any questions?

You can find us at
Twitters: @sindromequijote @Norman3D
Emails: mario.palmero@tequilaworks.com
norman3d@gmail.com