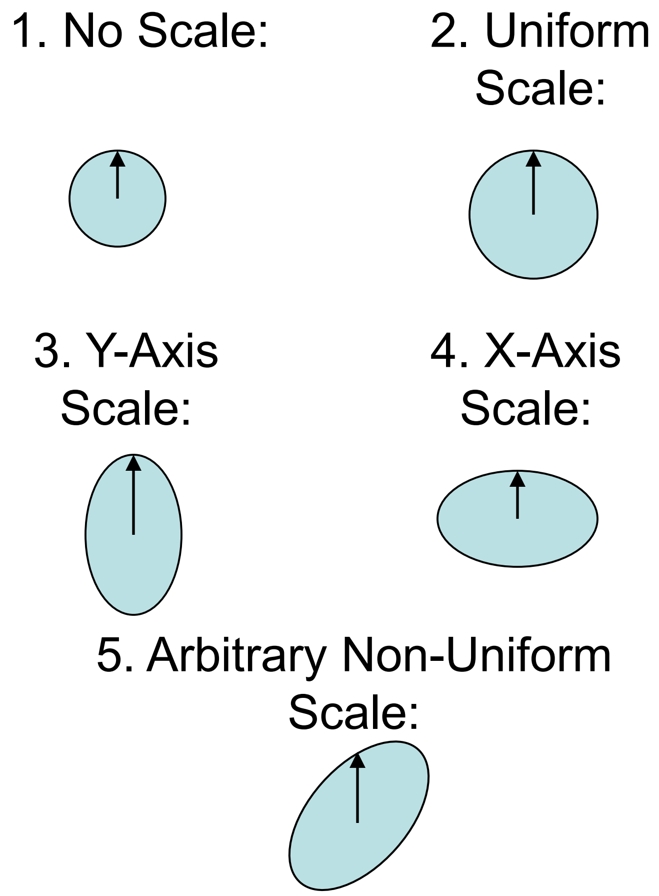
Non-Uniform Bone Scaling From Art Pipeline to Real-Time Rendering

Why Non-Uniform Bone Scaling?

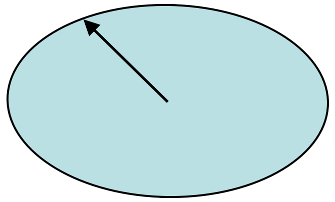
Different Forms of Scaling:

- The illustration on the right displays the different ways scales can be applied to a transformation matrix.
- Scaling illustrated in pictures 2-4 can be achieved using a ‘regular‘ transformation matrix with only ‘Position‘, ‘Rotate‘ and ‘Scale‘ components.
- Picture 5 illustrates a form of scaling that can only be achieved by introducing a ‘Stretch Quaternion‘ as a component to the transformation matrix.



Stretch Quaternion

- The stretch quaternion defines how the object is rotated before the scale component gets applied to the transformation matrix.
- The following illustration shows how a non-uniform scale is applied:
- 1. Apply inverse stretch quaternion
- 2. Apply scale
- 3. Apply stretch quaternion
- 4. Apply rotation
- 5. Apply position



How to Extract the Data

1. For each bone of each frame of animation in the 3D application (3DS Max, Maya, etc.), get the transformation matrix
2. Run decomp_affine ("Polar Matrix Decomposition" by Ken Shoemake, shoemake@graphics.cis.upenn.edu in "Graphics Gems IV") on these matrices and check if stretch quaternion is identity.
3. If stretch quaternion is not identity, mark the bone for this animation as non-uniformly scaled.
4. Store all frames in the animation as you would normally, but also store the stretch quaternion data for all non-uniformly scaled bones as marked in step 3.

How to Reconstruct the Data

- Code for recomposing of the final matrix from its components:

```
Matrix FinalMatrix;
if(CurrentBone.StretchQuat)
{
        Matrix PosMtx.FromTranslation(PosVector);          // sets 4th row
        Matrix ScaleMtx.FromScale(ScaleVector);            // sets scale entries
        Matrix RotMtx.FromQuaternion(RotQuat);             // quaternion to matrix
        Matrix StrMtx.FromQuaternion(StretchQuat);         // quaternion to matrix
        Matrix InvStrMtx.Inverse(StrMtx);                  // inverse of matrix
        FinalMatrix = InvStrMtx * ScaleMtx * StrMtx * RotMtx * PosMtx;
}
else
{
        Matrix PosMtx.FromTranslation(PosVector);          // sets 4th row
        Matrix ScaleMtx.FromScale(ScaleVector);            // sets scale entries
        Matrix RotMtx.FromQuaternion(RotQuat);             // quaternion to matrix
        FinalMatrix = ScaleMtx * RotMtx * PosMtx;
}
```

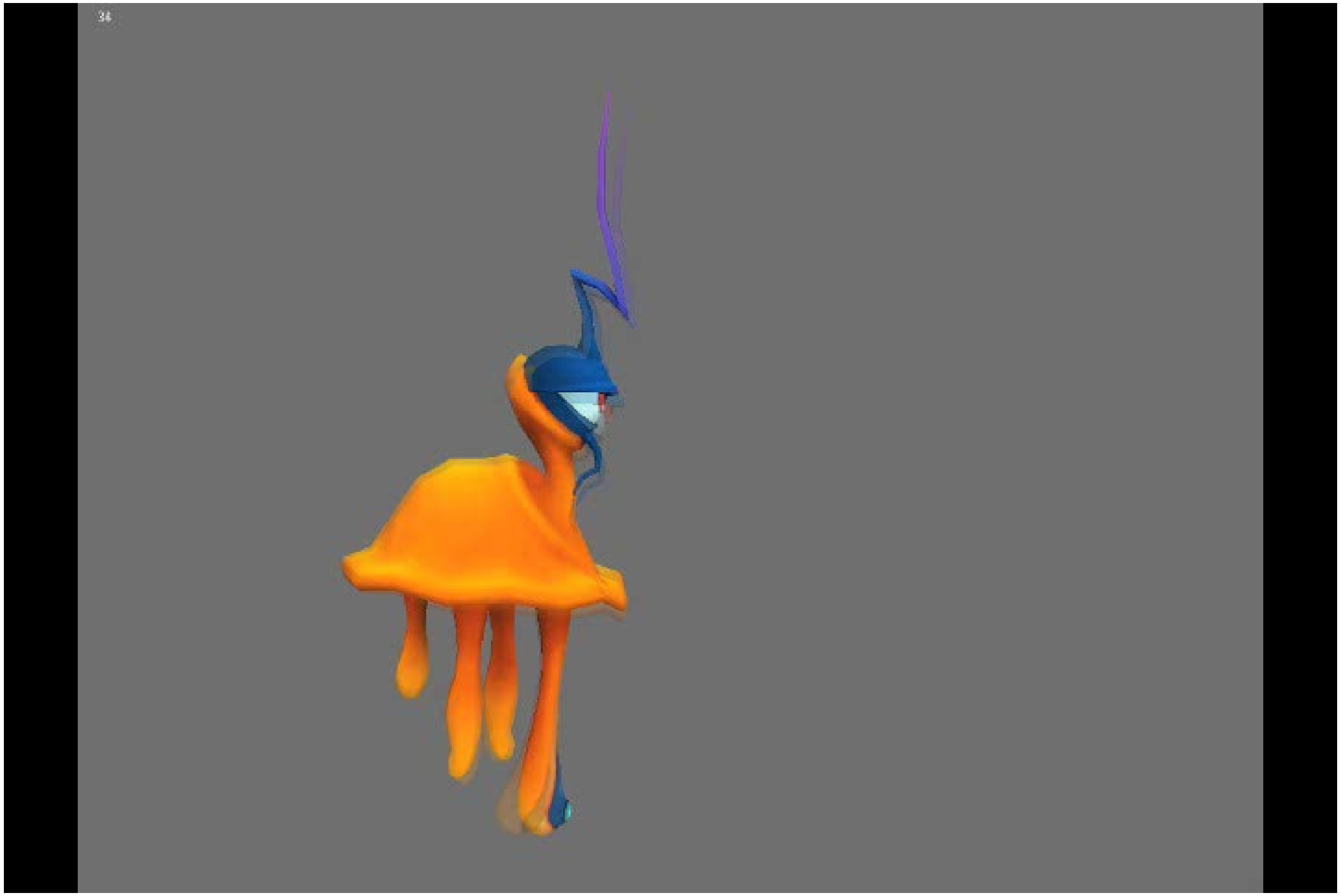
- As you can see, the performance is only impacted for bones that do stretch in a non-uniform way.
- Therefore, additional storage of the stretch quaternion component data is only required for these types of bones.

So, that‘s why!

Scale Inheritance Problem

- When traversing a skeleton, each child bone by default inherits the parent bones scale value. This can lead to problems when applying non-uniform bone scaling to a bone that has children. The children will most likely be affected by an undesired (sheared) scale component.
- 3D modelling programs such as 3DS Max, Maya, etc., therefore provide a per bone ‘Don‘t inherit scale‘ option.
- This data needs to be exported, stored with your animation data, and respected by the game engine for the animations to look correct.
- Not respecting these flags will result in wrongly sheared animations.

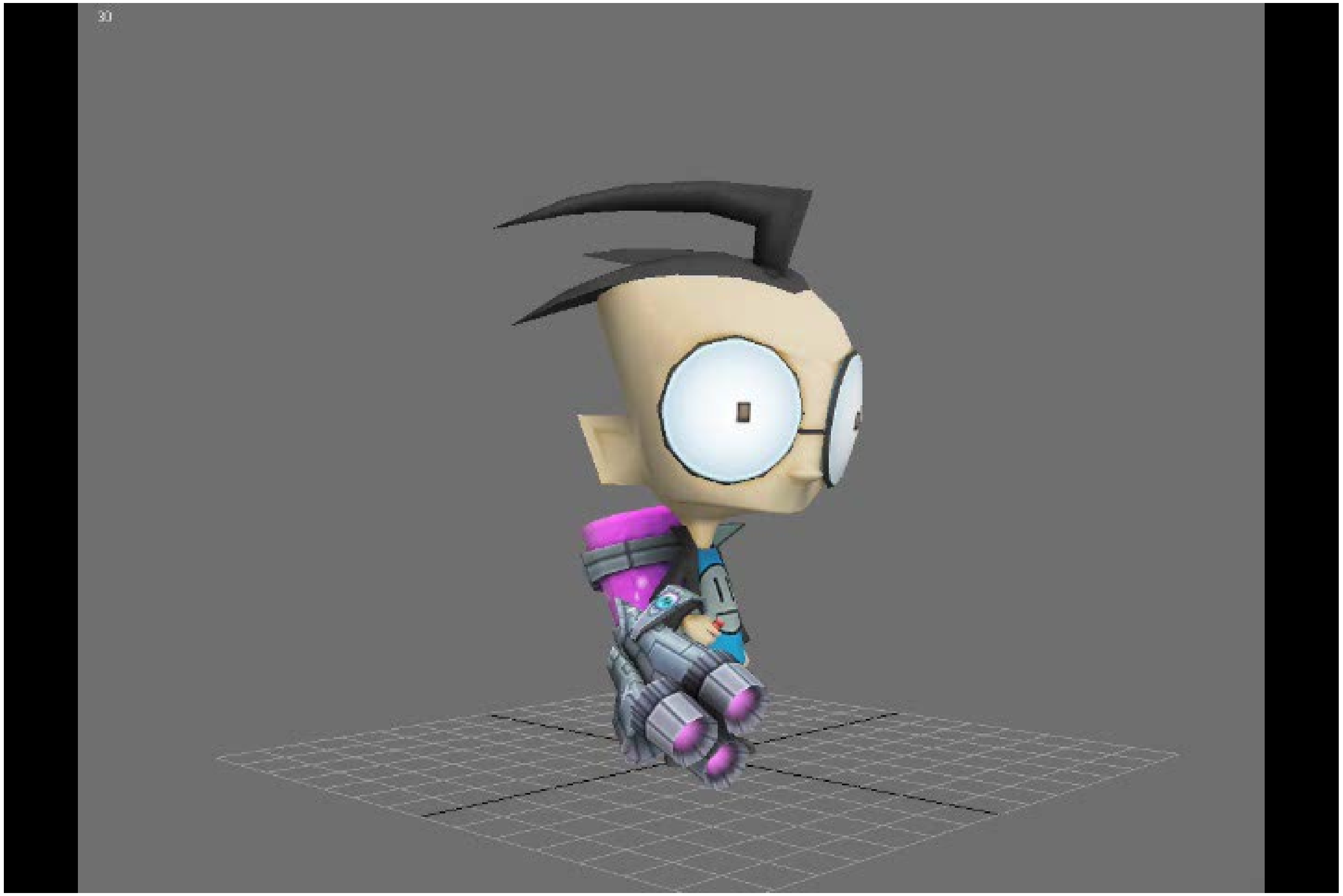
Scale Inheritance You Say?



Respecting Scale Inheritance Flag at Run-Time

- Traverse the skeleton and compute bone matrices using ONLY the position and rotation components.
- Store the resulting bone matrices twice.
- Traverse the skeleton again to compute and add the scale components to only the second set of matrices.
- Do a final pass on the scaled skeleton (second set of matrices) to compute the posed bone matrices with one rule: If the „Don‘t inherit scale“ flag is set, then look up the parent matrix from the unscaled set of matrices, otherwise use the parent matrix from the scaled version.

Aha! Scale Inheritance!



Thank you!

- Special thanks to the owners of Incinerator Studios and THQ for letting me use their content.
- Special thanks to Mike Howard and Nikita Wong for creating the animations.
- Special thanks to Michael Seare and Shaun McIntyre, my collaborators.
- Special thanks to Ken Shoemake for decomp_affine (Google it if you want to use it.)
- For questions I can be reached at: alex.ehrath@thq.com , alex@ehrath.org