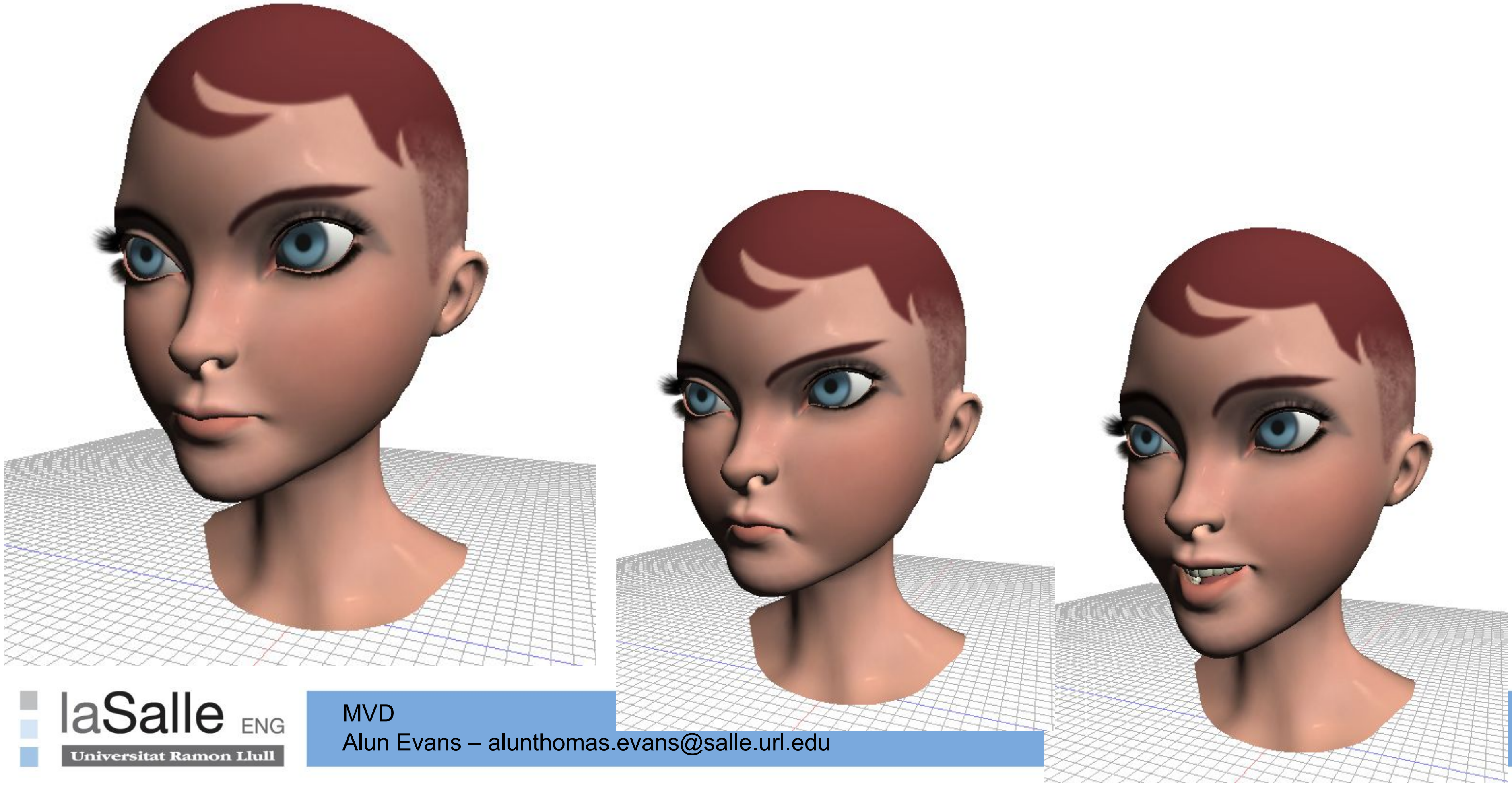
# MVD: Advanced Graphics 2

23 - Blend SHapes

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull

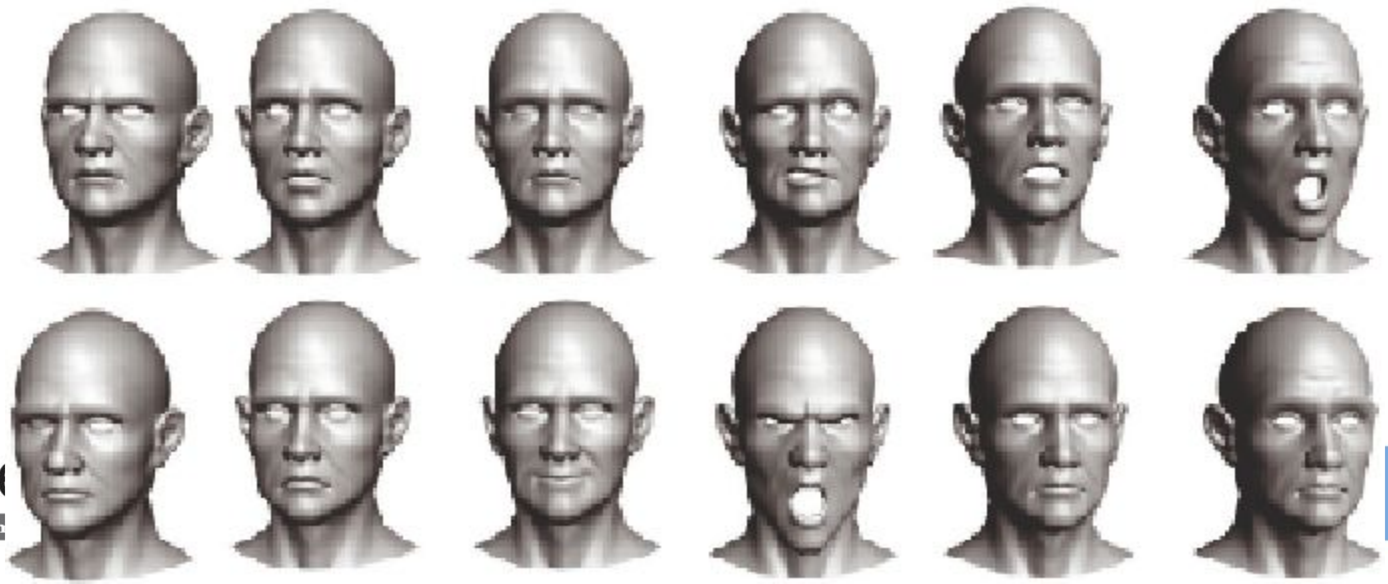
## Blend shapes aka Morph targets



## Blend Shapes

The key concept of blend shapes is each vertex of the base mesh is blended or ‘morphed’ to a different position.

This morphing is done by interpolating the original position of each vertex to the target position.



## Blend shapes pros

Not constrained by skeletons. Don’t have to worry about potential errors in skin-binding.

Artist has much more control of final animation (an old-name used to be ‘per-vertex’ animation.

Makes is particularly useful for fine animation such as skin, faces, cloth etc.

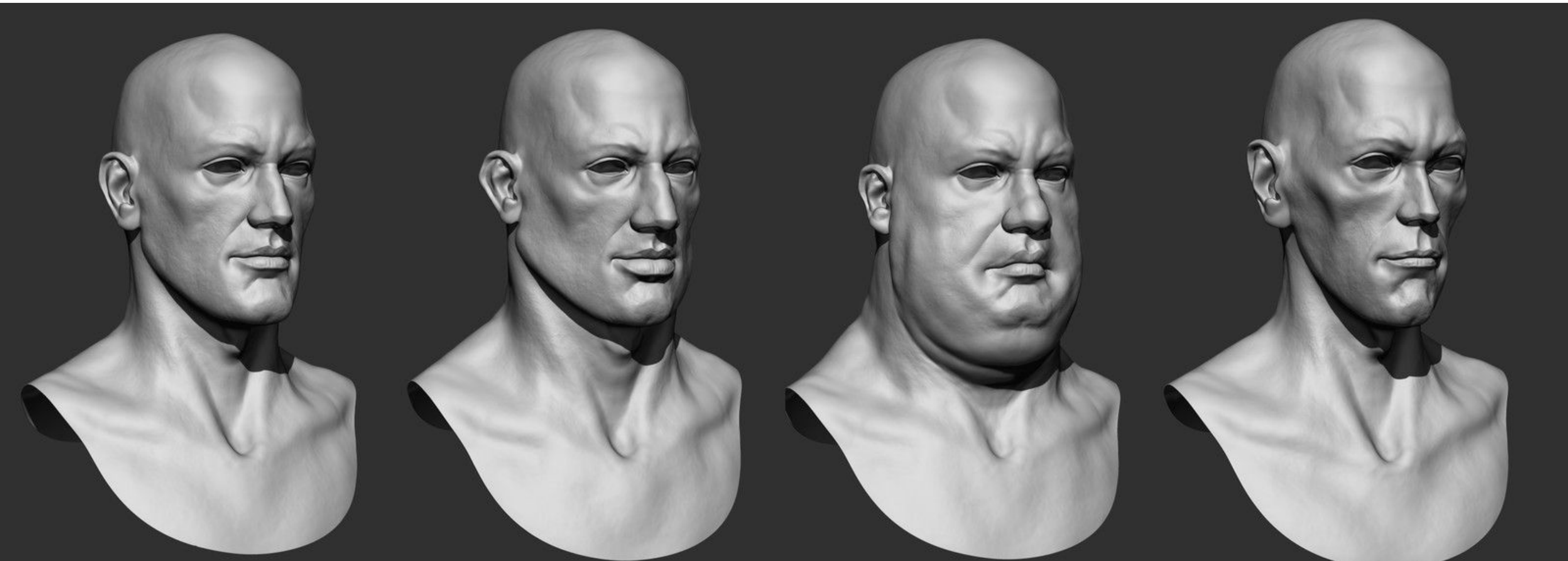
Can blend multiple targets to get ‘indirect targets’.

## Blend shapes cons

Requires deformation of each vertex = very labour intensive

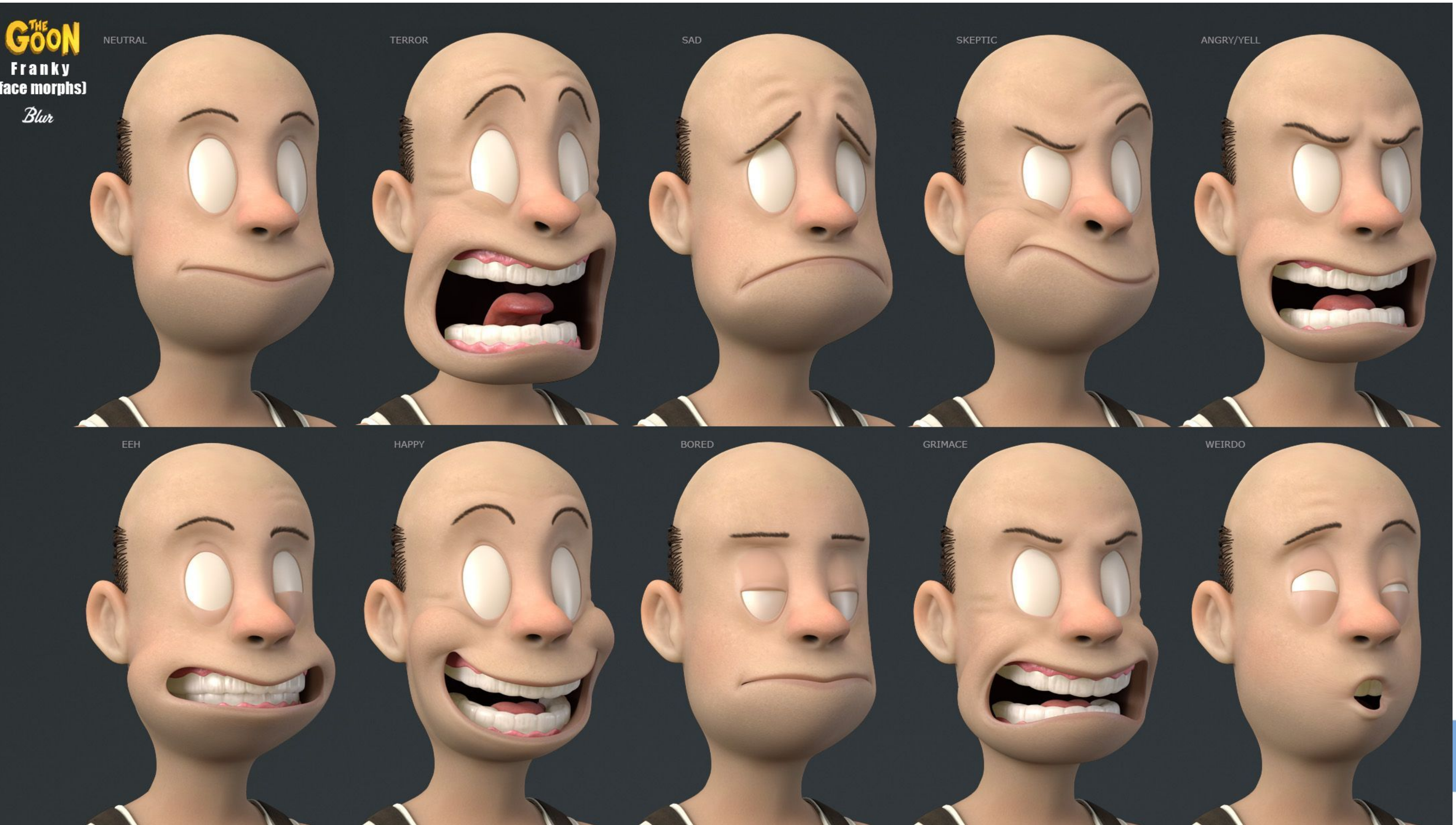
Can’t reuse work to create other animations (like you can with rigged mesh)

Also, if targets are interpolated too quickly, the effect can be very ‘shaky’ - only really a problem if frame-rate is low though



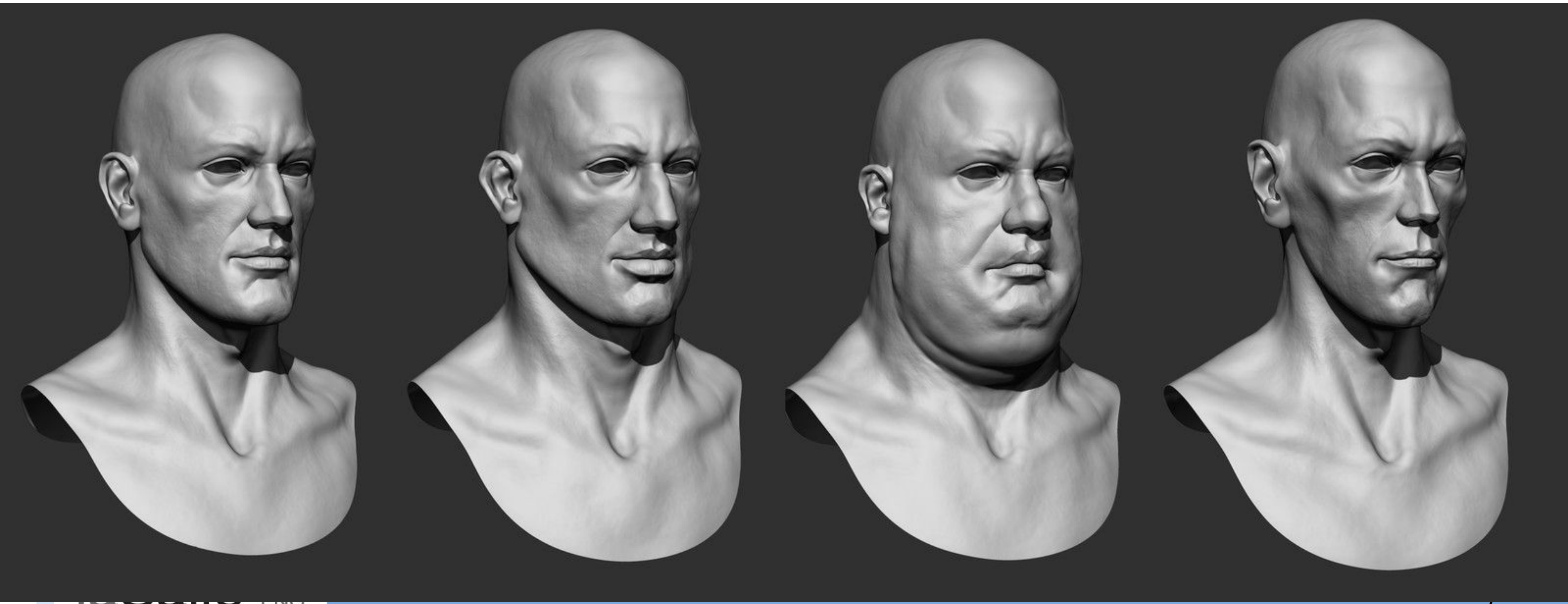
## Blend shape usage

Typically facial animation



## Blend shape usage

But also, character creation

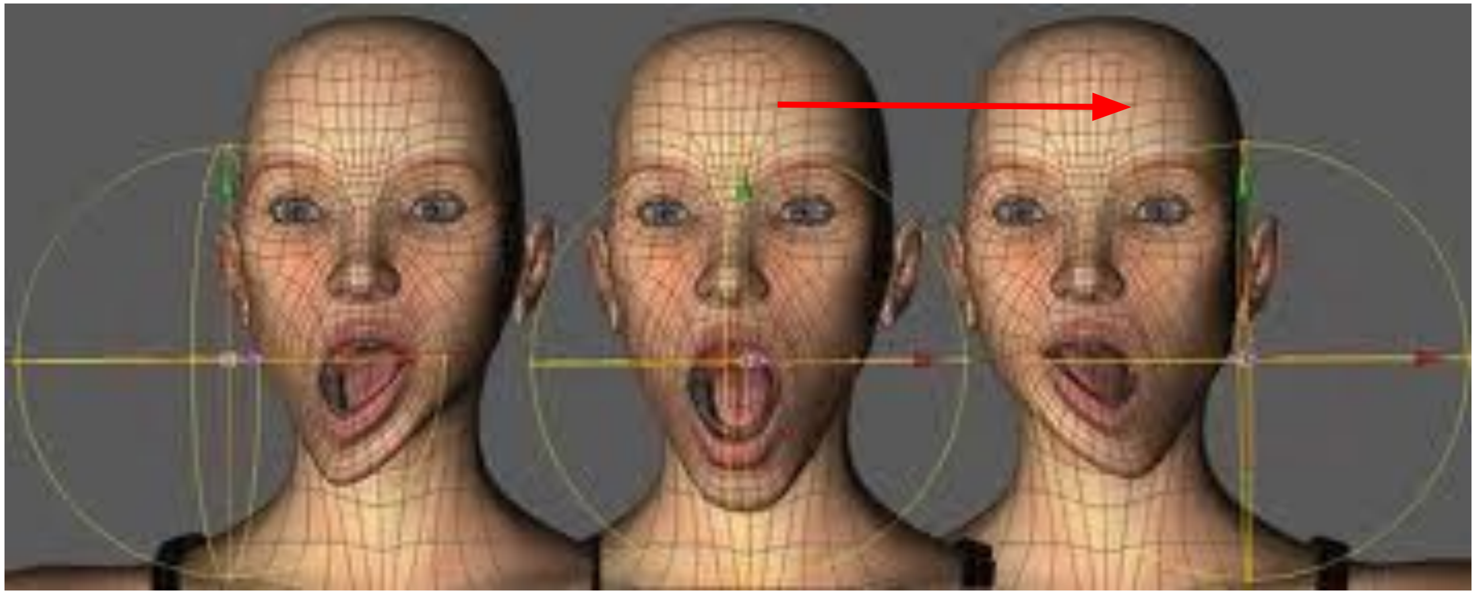


## Blend shape usage

Body animation - only in older games where there were hardly any vertices to animate!

## Morph target: 1 - 1 equivalence

All targets must have *exactly* the same geometry. The whole point is that there is an offset between each vertex



## Calculating the offset

We calculate the offset per vertex like this:

$$\mathtt{Offset}_v = \mathtt{Blend\_Shape}_v - \mathtt{Original\_Shape}_v$$

Then the final vertex position is

$$\mathtt{Final\_Position}_v = \mathtt{Original\_Shape}_v + \mathtt{Offset}_v * \mathtt{blend\_weight}$$

where $0 < \text{blend\_weight} < 1$

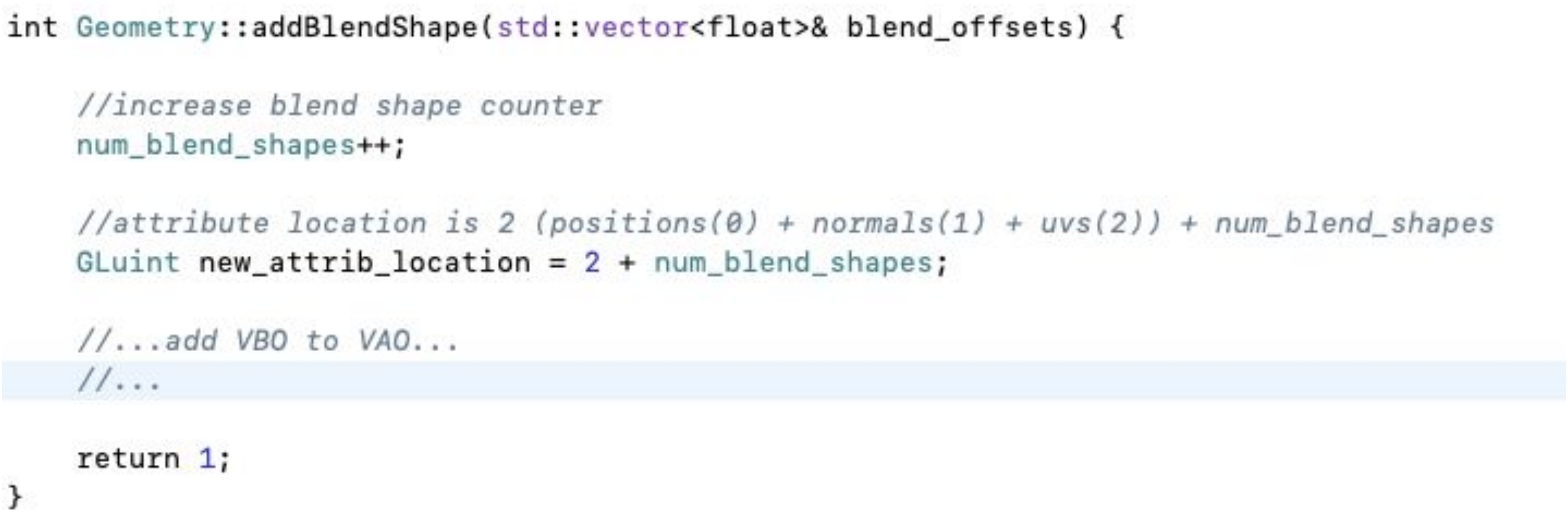
## Reading Blendshape geometry

```
//add first blend shape
Parsers::parseOBJ("data/assets/toon/toon_happy.obj", vertices, normals, uvs, indices);
toon_geom.addBlendShape(vertices);
```

We only need to ‘keep’ the vertices of the geometry

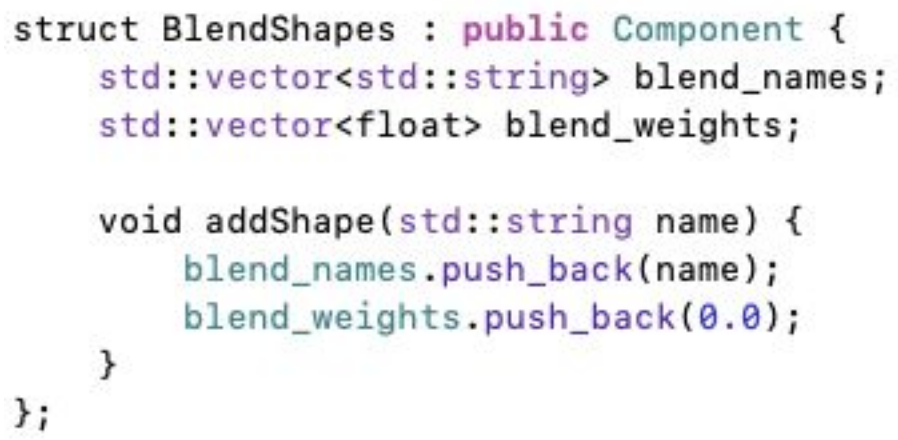
Easiest – just call ParseOBJ and pass the vertices array to a new function

## Task adding blend shape to geometry



```
int Geometry::addBlendShape(std::vector<float>& blend_offsets) {

    //increase blend shape counter
    num_blend_shapes++;

    //attribute location is 2 (positions(0) + normals(1) + uvs(2)) + num_blend_shapes
    GLuint new_attrib_location = 2 + num_blend_shapes;

    //...add VBO to VAO...
    //...

    return 1;
}
```

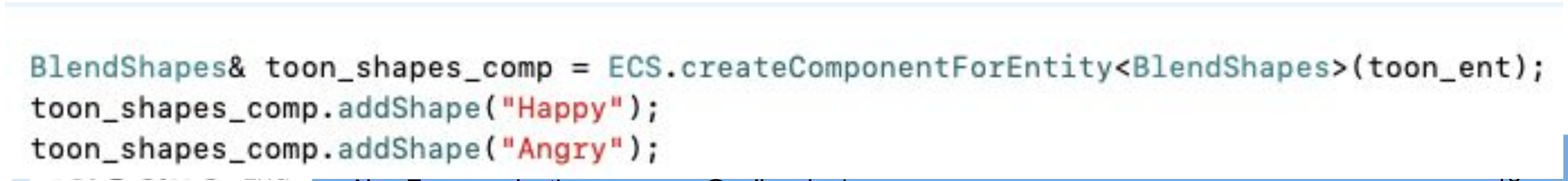
## Controlling Blend Shapes



```
struct BlendShapes : public Component {
    std::vector<std::string> blend_names;
    std::vector<float> blend_weights;

    void addShape(std::string name) {
        blend_names.push_back(name);
        blend_weights.push_back(0.0);
    }
};
```

Create a component that that stores names of blend shape and current interpolation values, in two arrays

The order of adding shapes to these arrays must be the same order as adding the geometry buffers!



```
BlendShapes& toon_shapes_comp = ECS.createComponentForEntity<BlendShapes>(toon_ent);
toon_shapes_comp.addShape("Happy");
toon_shapes_comp.addShape("Angry");
```

## Task: The shader

Add attributes for blend shapes

Add uniform float array `u_blend_weights`

Calculate offset from blend target to base position

Modified vertex = current vertex +

```
foreach target {
    offset * blend weight
}
```

**Test by hard-coding weights in shader**

## Task: Set Uniforms in renderMeshComponent

Given you function Shader::setUniformFloatArray

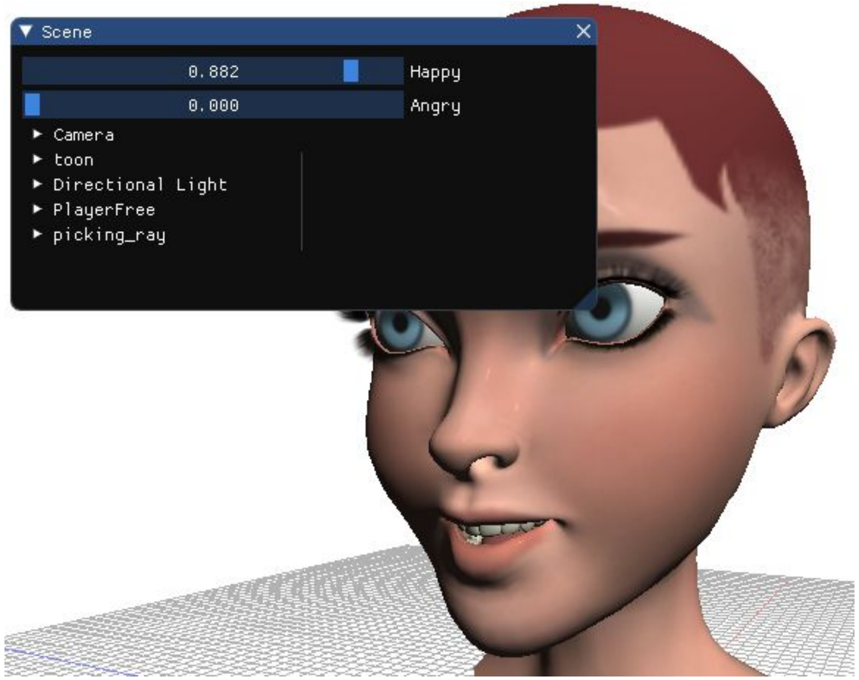
Receives:

- enum of uniform (U_BLEND_WEIGHTS)
- pointer to start of weights array (stored in Component)
- size of weights array (stored in Component)

## Task: Changing weights in real-time

Use Dear imGUI

ImGui::SliderFloat(
- string name
- &float_value_to_change
- min
- max
- decimal places



## Advanced Task:Normals

Modify code to include variations in normals!