

Procedural Animation

Particles Trajectories

Particles Systems History

2/81

One of the first animated model in CG



[Particle systems - A Technique for Modeling a Class of Fuzzy Objects, William T. Reeves, Lucasfilm, 1983 (Star Trek II)]



[Karl Sims, Particle dreams, 1988]

Example of particle system

3/81

Free fall of spheres under gravity

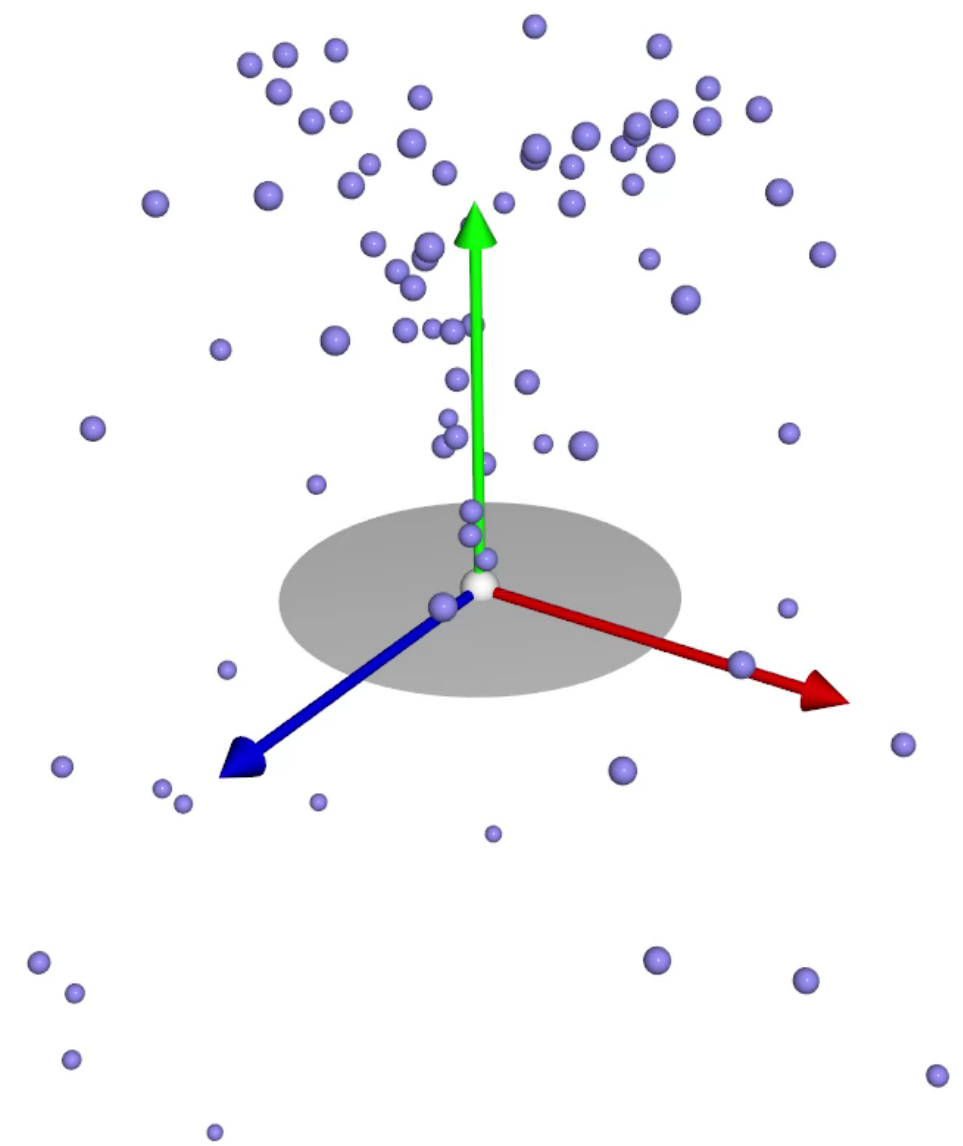
- Geometrical representation of each particle: sphere
- Equation of motion $p(t) = \frac{1}{2}gt^2 + v_0t + p_0$
- Initial position and speed may be placed at random position
- Each particle may have a different life time

Q. What are the parameters used for p_0 and v_0 in this example ?

$p_0 = \dots$

$v_0 = \dots$

Note: Coordinate system $(x, y, z) = (\text{red}, \text{green}, \text{blue})$.



Bouncing spheres

4/81

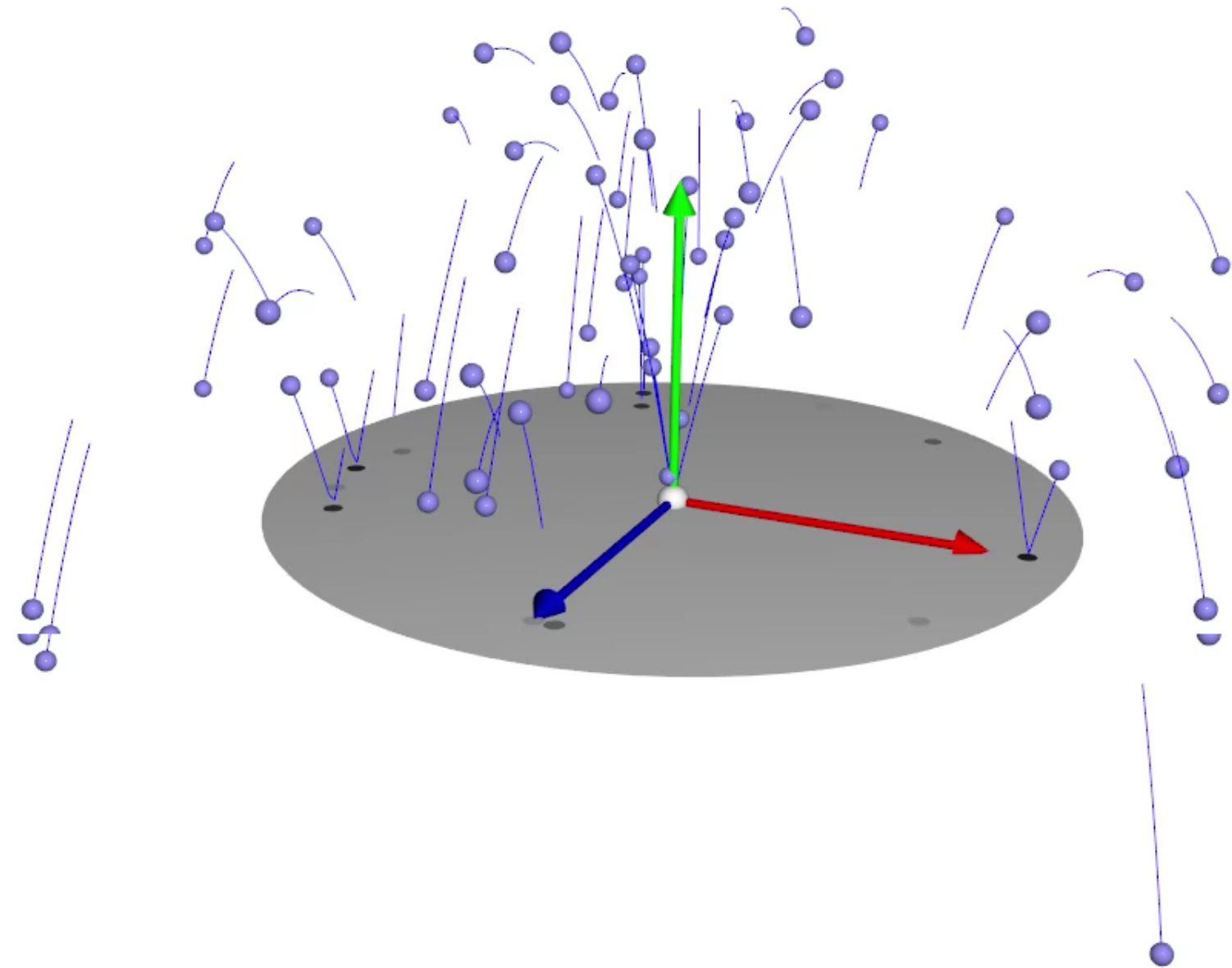
Q. What is the equation of motion ?
(taking into account the bouncing)

$$p(t) = \dots$$

Help:

- Consider a particle emitted at time $t = 0$
- At what time t_i , the particle touches the ground ?
- What is the new speed after impact ?
- Express the complete equation of trajectory
(piecewise definition)

Bonus: How the "impact shadow" effect
is computed ?



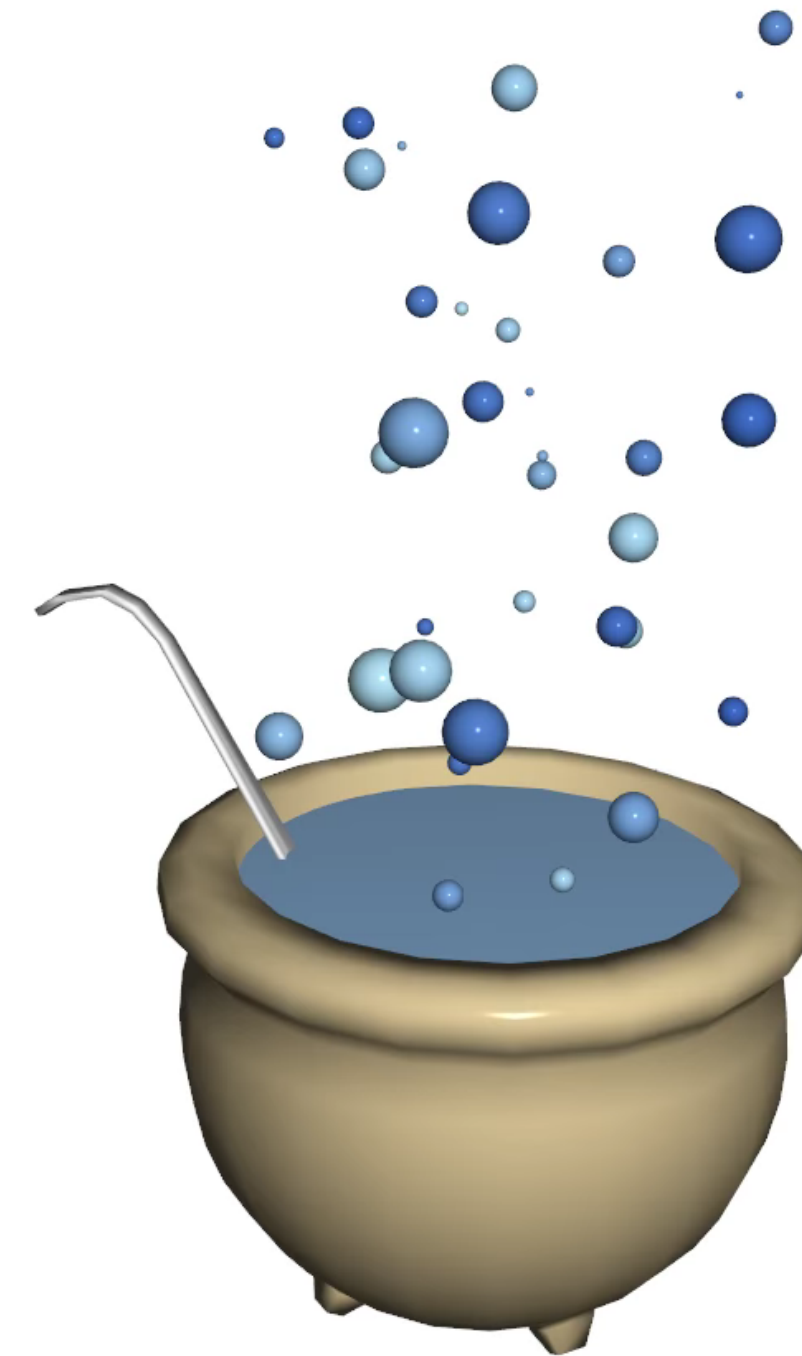
Manually defined/arbitrary motions

5/81

Trajectories are not restricted to physics-based equations

Q. *What are the parameters (and initial value) associated to each particle ?*

- position = ...
 - radius = rand([0,0.1])
 - ...
- *What is the equation of motion of a particle ?*
- $p(t) = \dots$



Billboards, Impostors, Sprites

6/81

Particles can be displayed as small images/thumbnails (instead of simple sphere)

In practice

- Each particle is displayed as a quadrangle
- A texture is mapped on the quad
- The texture can contain transparency : quad geometry is invisible
 - Fake a more complex geometry
- The quad can be facing the camera at all time (billboard)
- The texture can be animated (sprite)
- Texture can be adapted to the point of view (impostors)

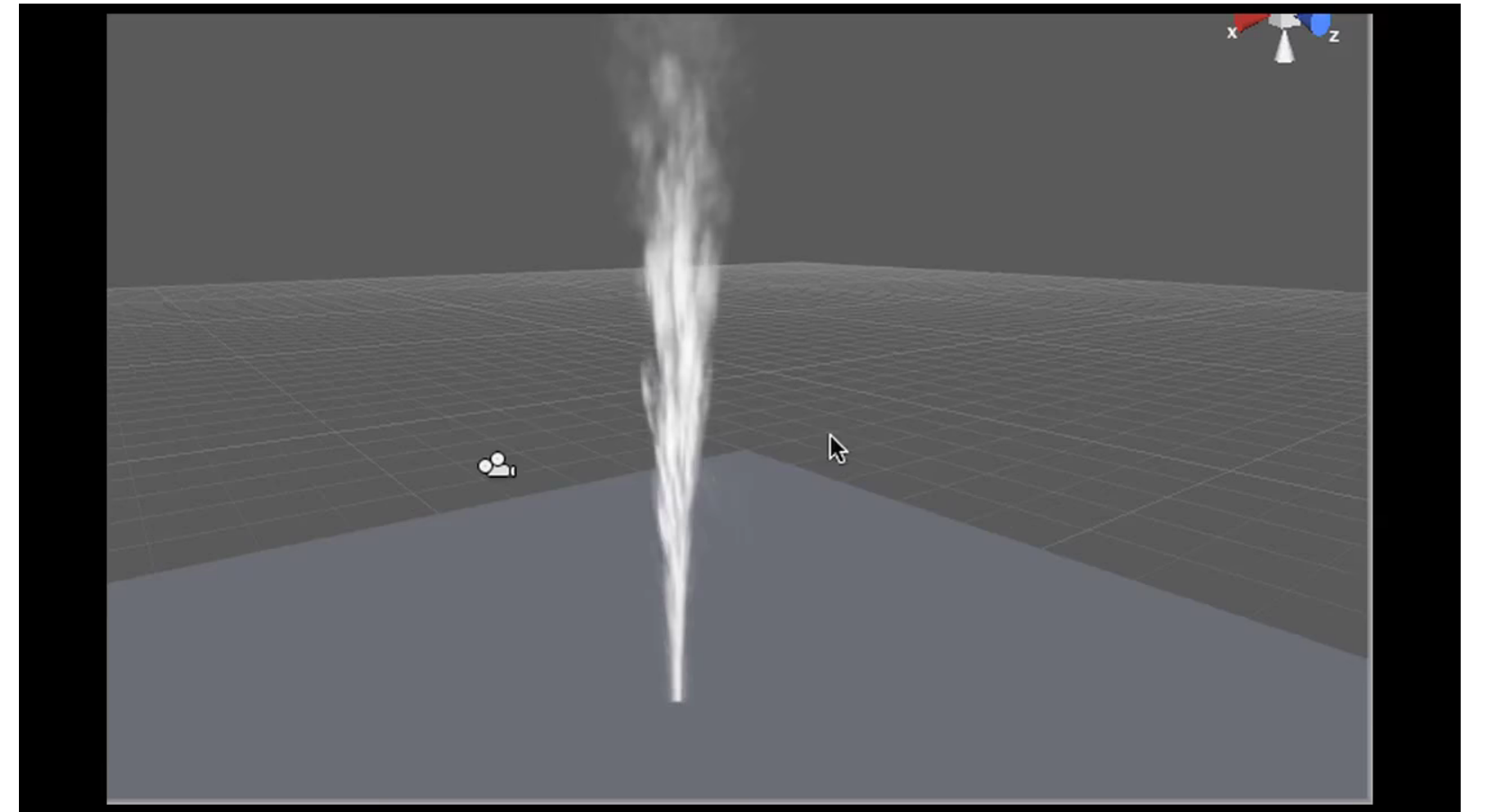
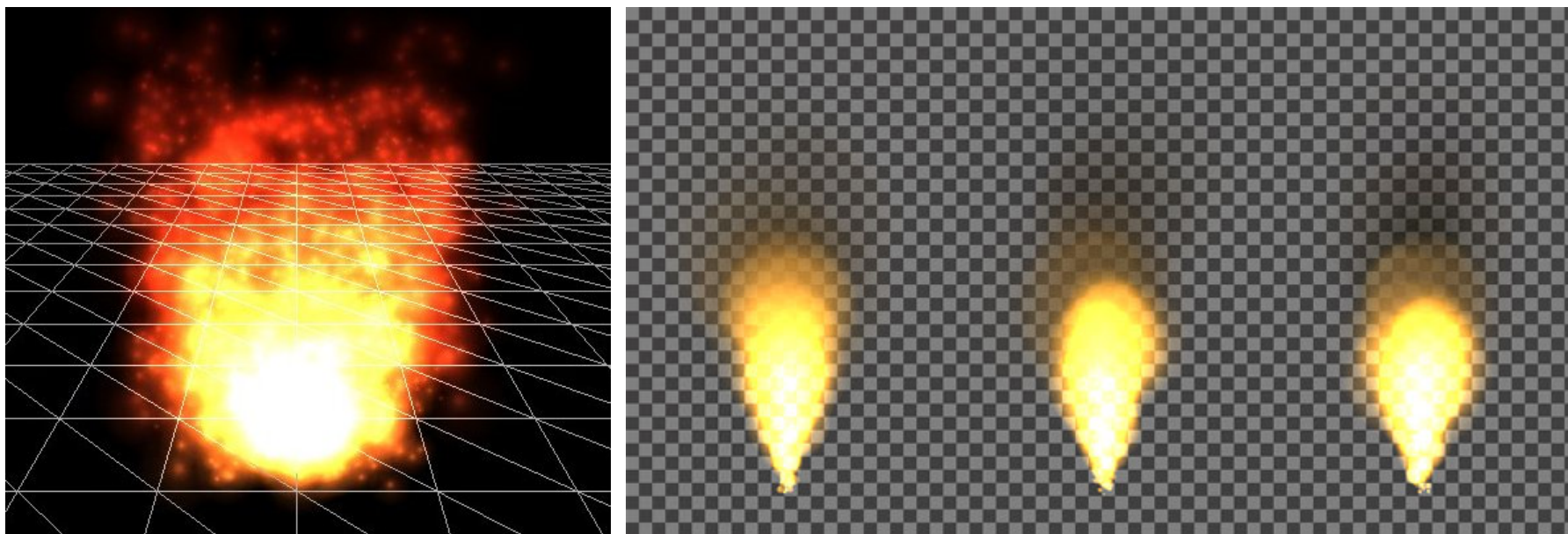
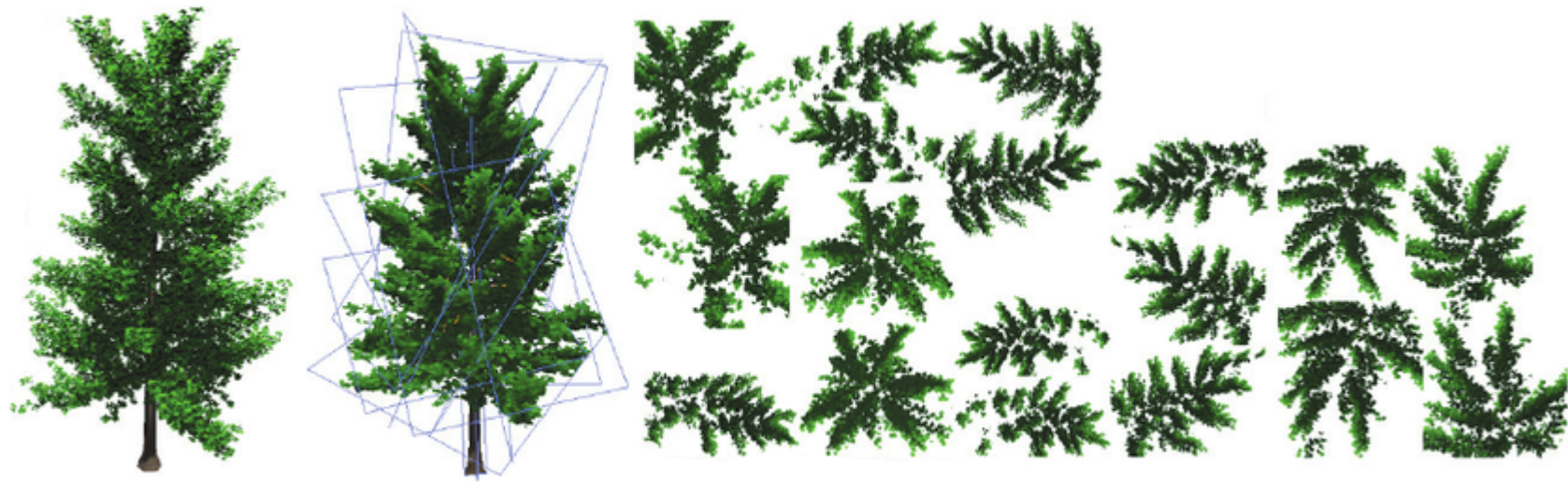


Usage of billboards

7/81

Large use of billboards for complex models

Vegetation, fire, smoke, etc.



Example of billboard

8/81

Follow up of the example

Q. How to model the following animation ?



Use case in production

9/81

How to model the *horse heads made of water* ?

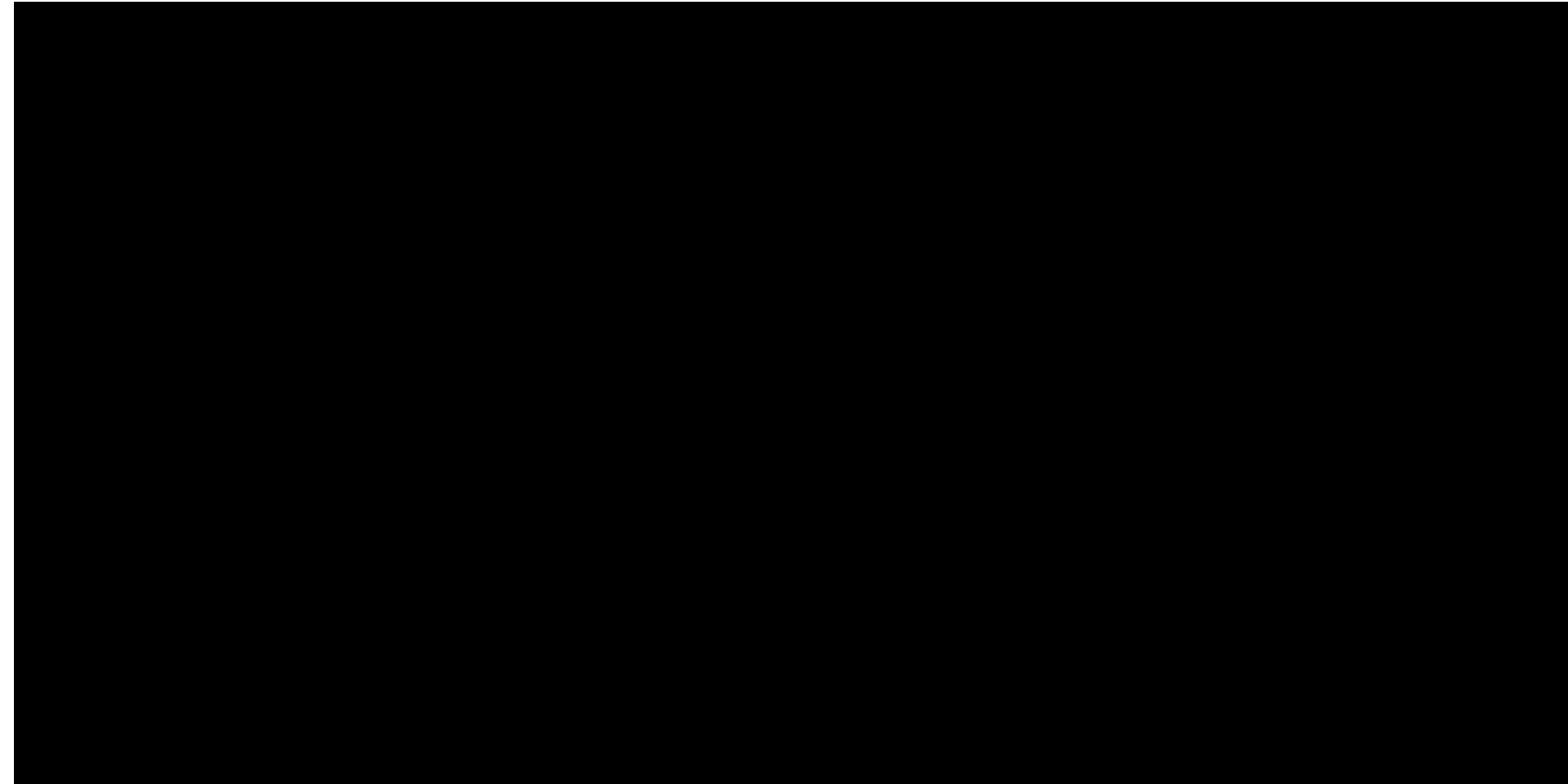


The Lord of the Rings

Use case in production

10/81

How to model the *horse heads made of water* ?

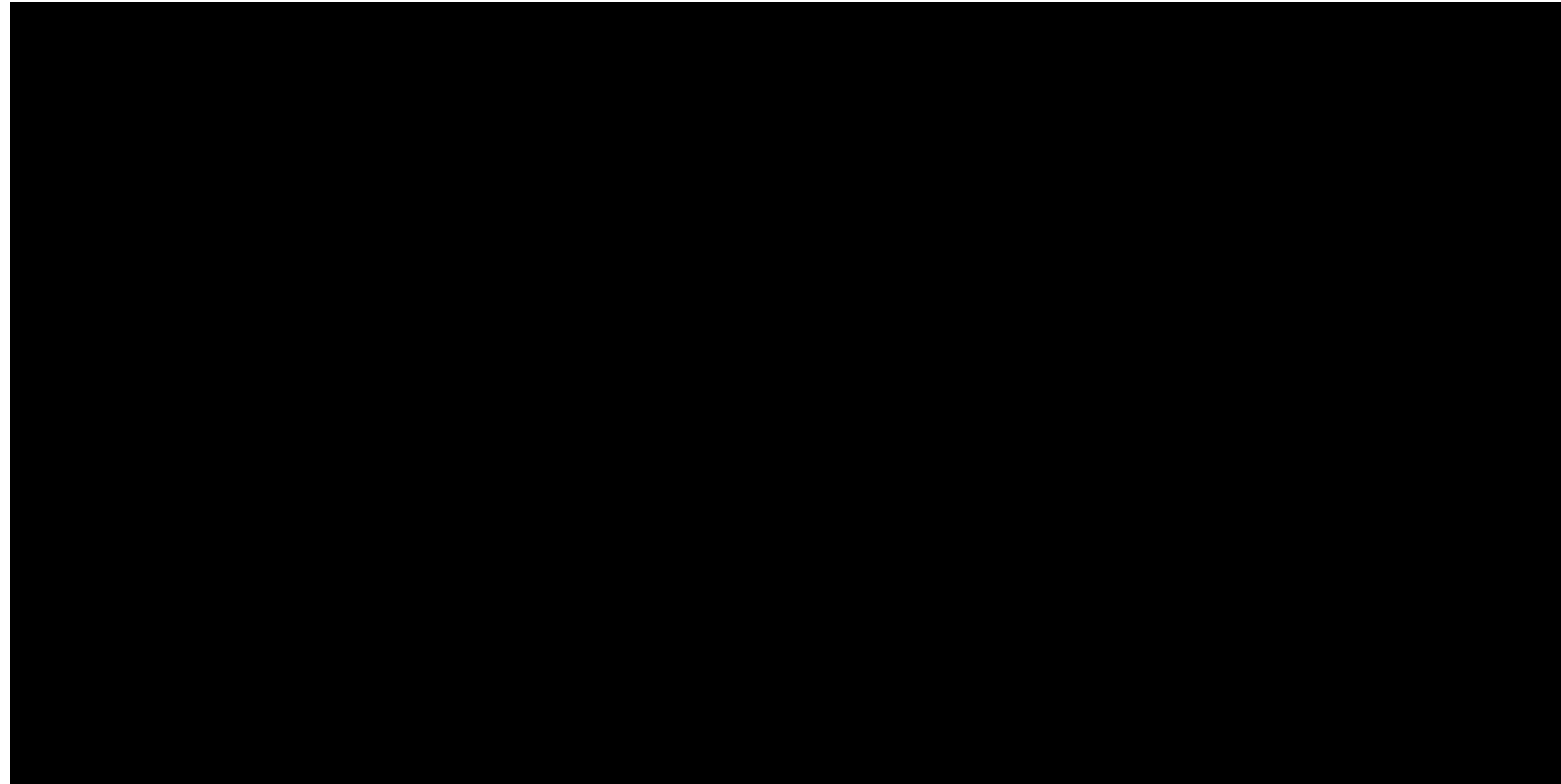


The Lord of the Rings

Use case in production

11/81

Full Making-Of



The Lord of the Rings

Implementation - transparency in OpenGL

12/81

Transparency is not straightforward using projection/rasterization rendering.

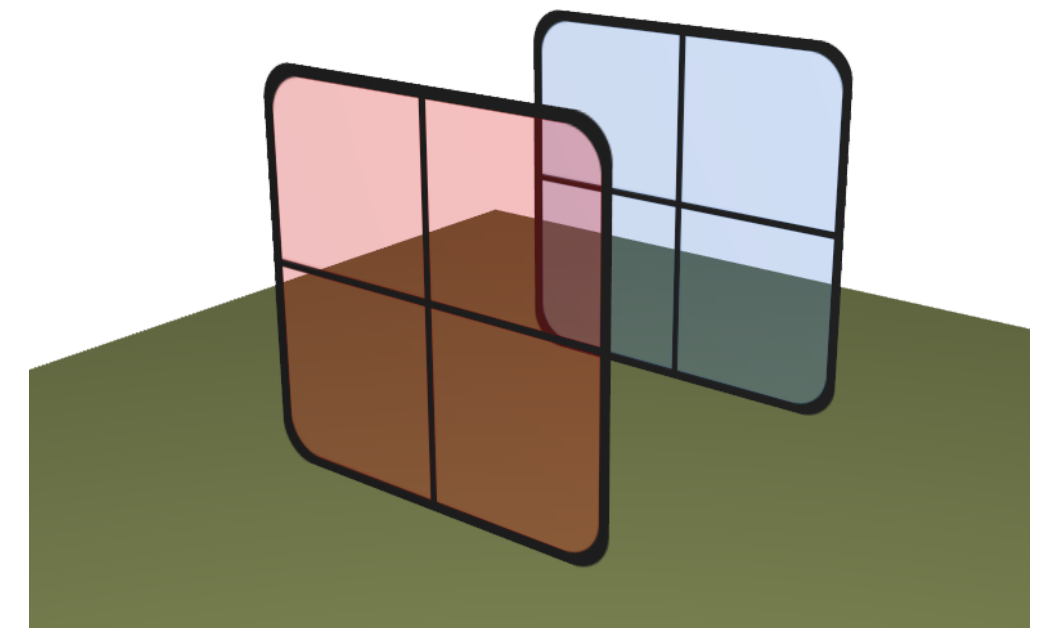
No perfect solution

⇒ Possibility to blend color for each pixel using α channel (rgba)

Need to be explicitly activated in OpenGL

```
glEnable(GL_BLEND);  
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);  
... Draw Calls ...  
glDisable(GL_BLEND);
```

- Various blending are possible for multiple effects (see [glBlendFunc](#))
- For transparency: (*GL_SRC_ALPHA*, *GL_ONE_MINUS_SRC_ALPHA*)
⇒ $\text{newColor} = \alpha \text{ currentColor} + (1 - \alpha) \text{ previousColor}$



Important note:

The **order** of the draw calls is important (previousColor/currentColor)

The depth-buffer cannot handle the notion of transparency

Implementation - transparency in OpenGL

13/81

General Process:

- 1. Draw all **non-transparent** objects
- 2.a Activate color blending, deactivate depth buffer write
- 2.b **Sort** transparent objects by depth to the camera
$$p_{proj} = Proj \times ModelView \times p$$
$$z_{depth} = p_{proj}.z / p_{proj}.w$$
- 3. **Draw transparent objects** from furthest to closest

```
... Draw all non-transparent objects ...
glEnable(GL_BLEND); // Color Blending
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glDepthMask(false); // do not write on depth buffer
... Sort transparent objects by depth ...
... Draw all transparent objects from furthest to closest ...
glDisable(GL_BLEND);
// do not forget to re-activate depth buffer write
glDepthMask(true);
```

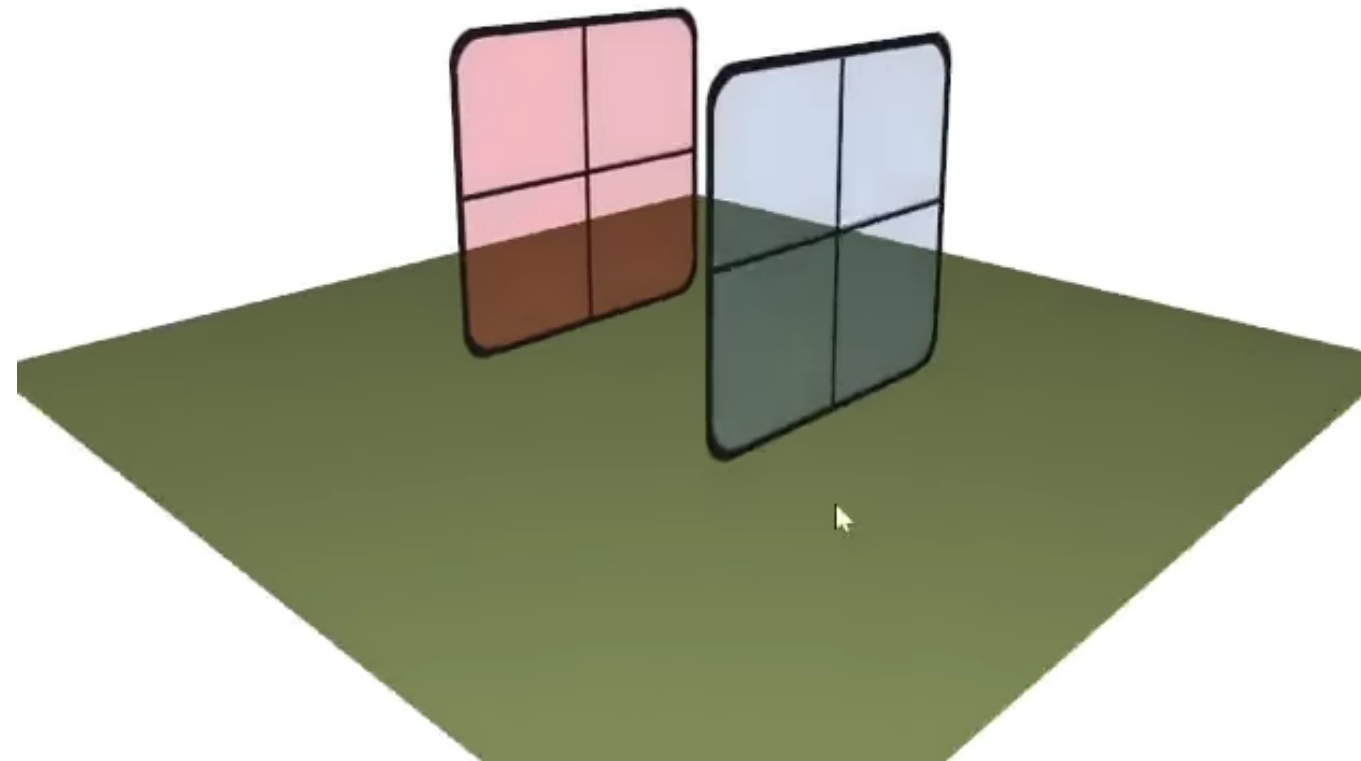
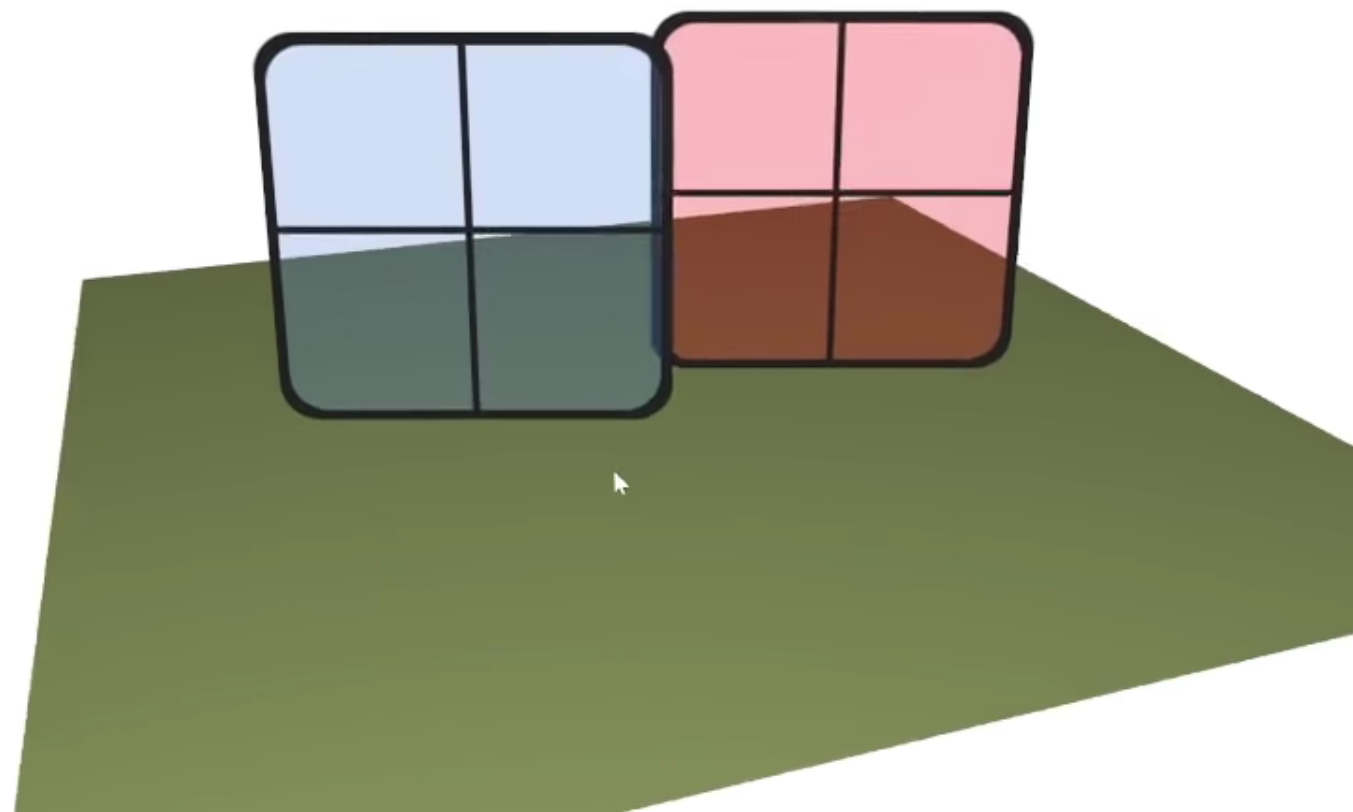


Image Blending - common errors

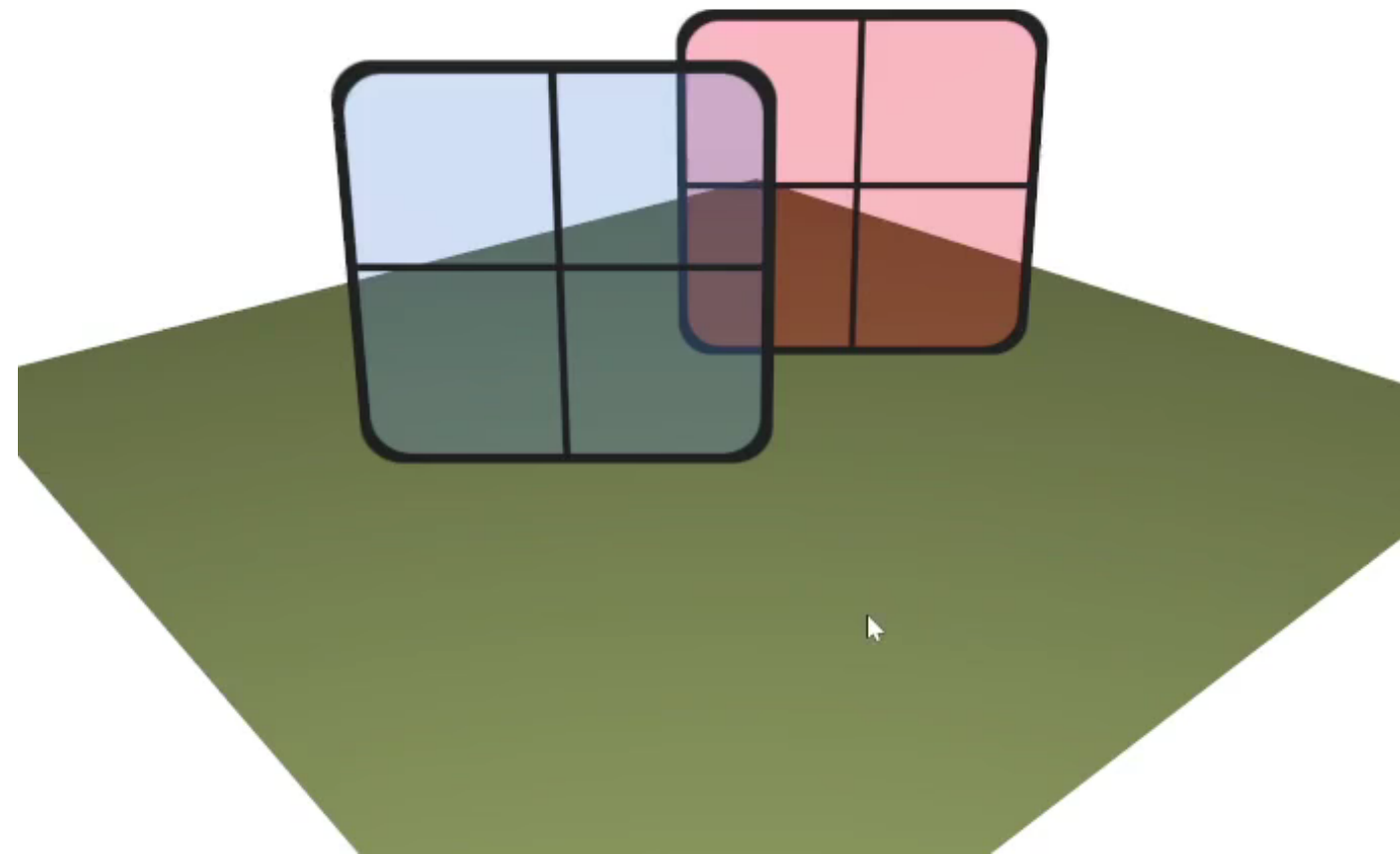
14/81

Keeping depth-buffer write activated
(without sorting)



Strong artifact, cannot be used

No sorting, depth buffer write de-activated



Note:

Can be OK for blurry sprites (like smokes)
where element order is hard to distinguish.
Avoids the cost of the sorting.

Implementation - transparency in OpenGL

15/81

For impostors shape with fully opaque/transparent fragment:

discard can be used on transparent fragments

discard = Stop fragment shader execution (nothing is written)

(+) Generic, standard display

no need to sort shapes, depth-buffer works

(-) Possible aliasing artifacts

(-) Cannot be used on semi-transparent objects



In fragment shader:

```
float limit = 0.6;  
if( texture.a < limit) {  
    discard;  
}
```

Instancing

16/81

In particle system we need to display lots of similar shapes

Multiple draw calls ($N > 1000$) can be costly

each draw call requires synchronization b/w CPU-GPU

```
// Slow for large N
for(int k=0; k<N; ++k)
    someOperator(k)
    draw(shape) // ... glDrawElements(...)
```

Solution: Use Instancing

glDrawArrays → [glDrawArraysInstanced](#)

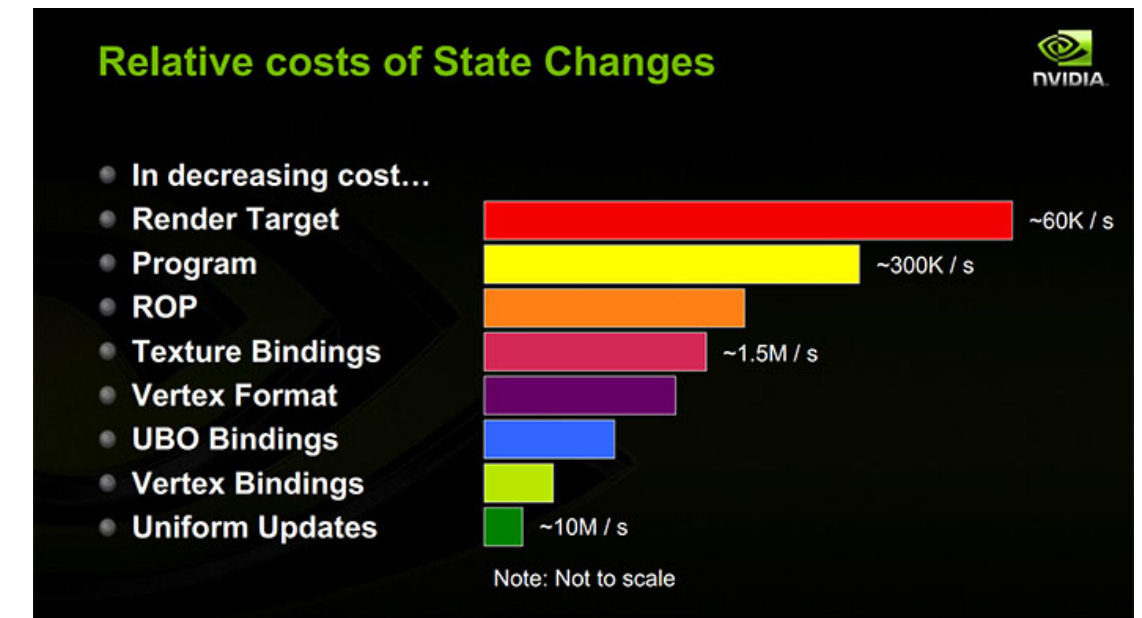
glDrawElements → [glDrawElementsInstanced](#)

In the C++ code

```
// 1 unique call, N is a parameter
drawInstanced(shape, N) // ... glDrawElementsInstanced(...,N)
// No more explicit loop
```

In the Shader:

```
// ...
someOperator(gl_InstanceID)
// ...
```



[NVIDIA]

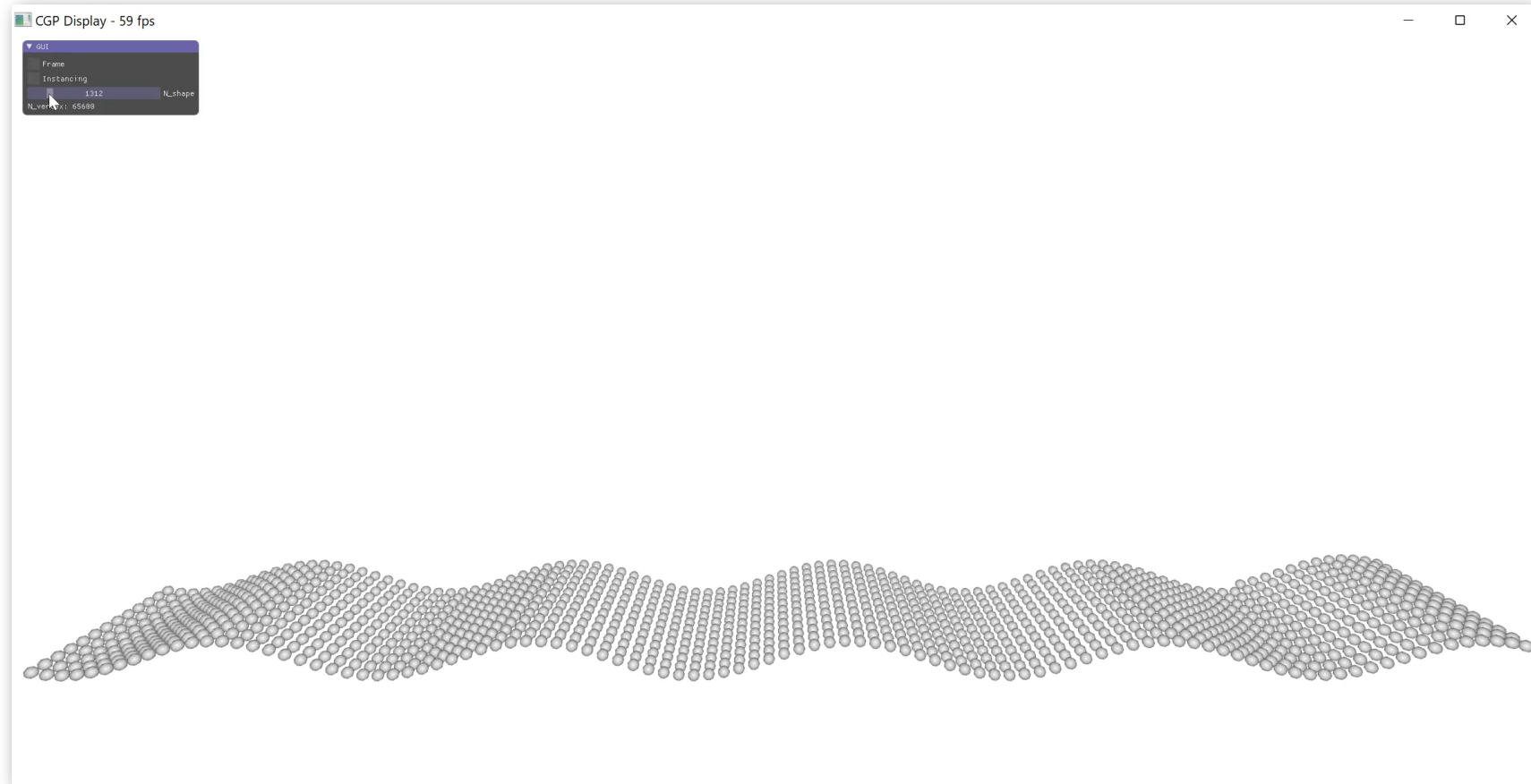
(+) Full capability of your GPU

(-) Require a specific shader

Instancing Demo

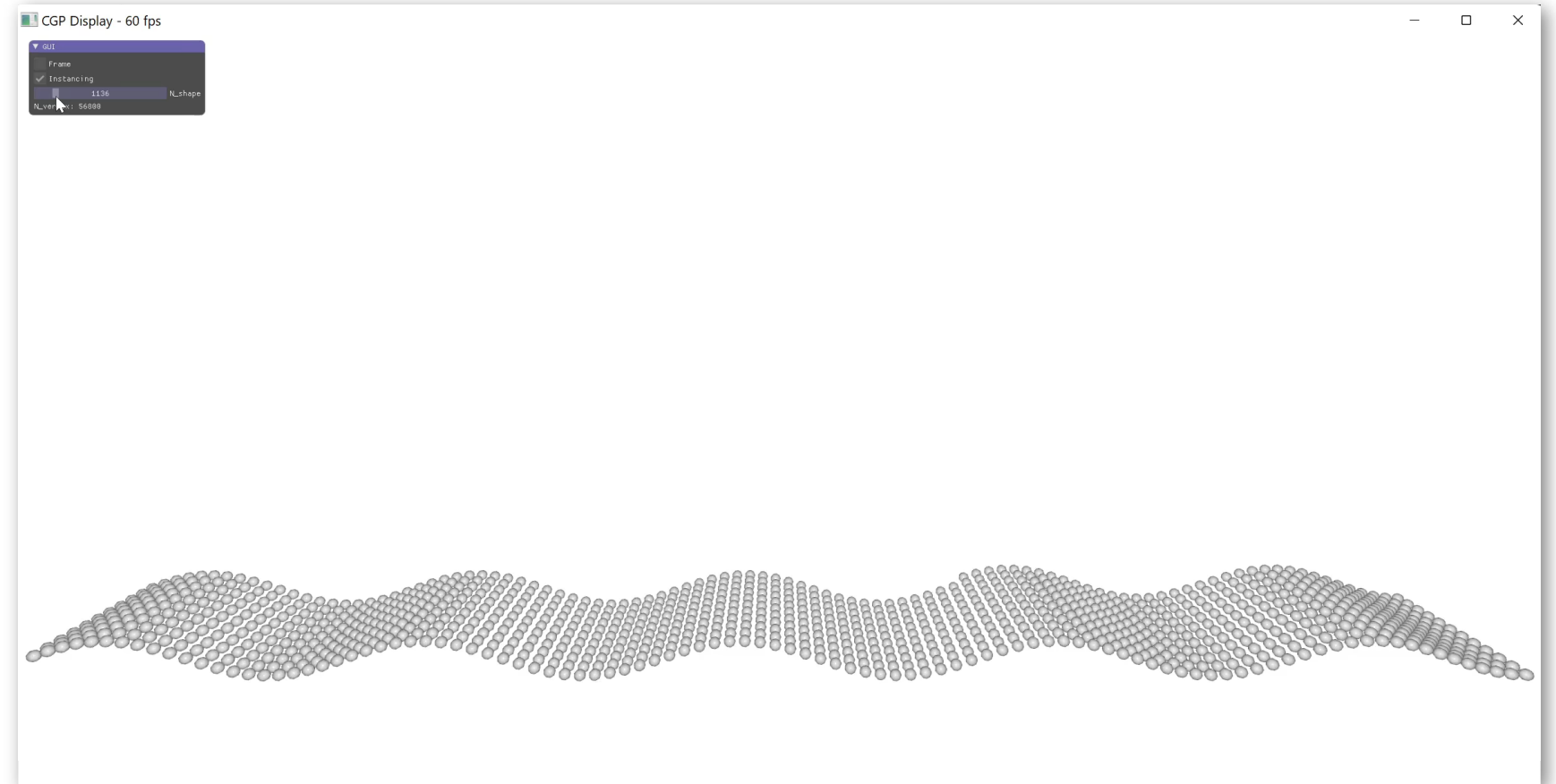
17/81

Without Instancing



Slow for 15,000 particles
#vertices=750,000

With Instancing



Can display 450,000 particles at 60fps (depends on GPU)
#vertices=22,000,000

Perlin Noise Terrain

18/81

Consider the surface S defined as

$$S(u, v) = \begin{cases} x(u, v) = u \\ y(u, v) = v \\ z(u, v) = h P(s(u + o), s(v + o)) \end{cases}$$

The perlin noise

$$P(u, v) = \sum_{k=0}^N \alpha^k f(2^k u, 2^k v)$$

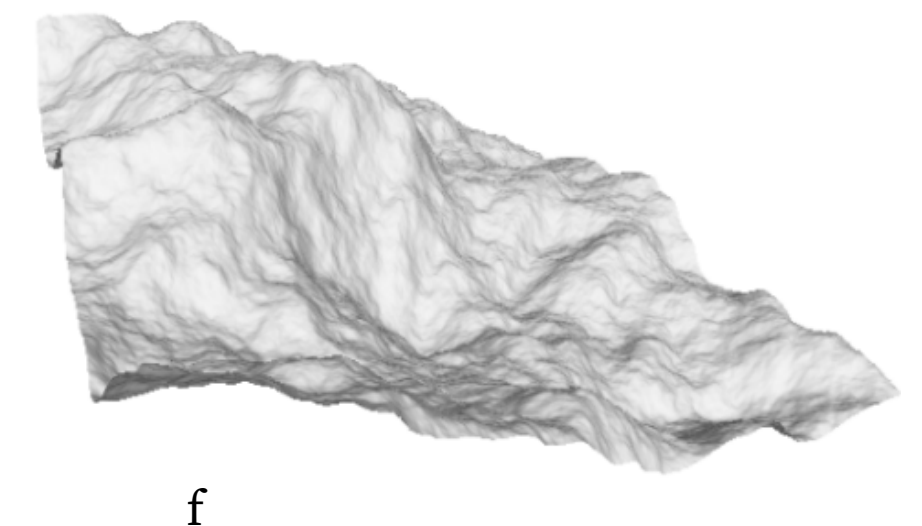
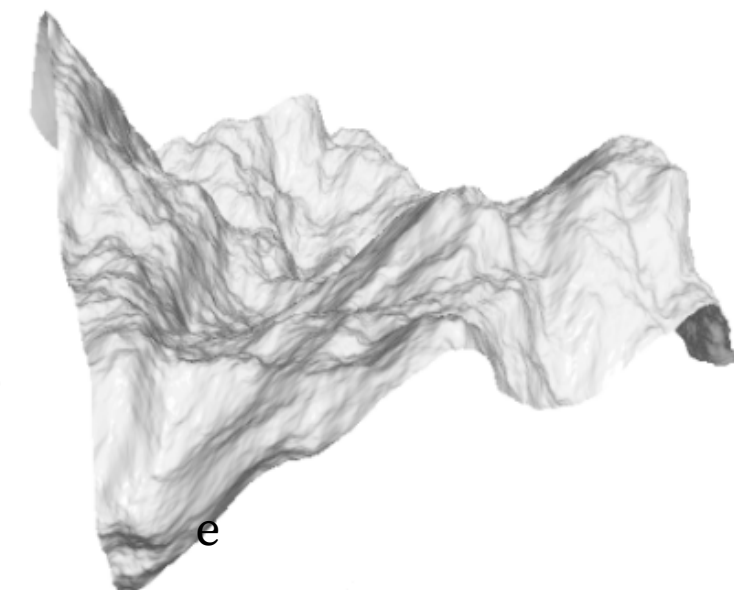
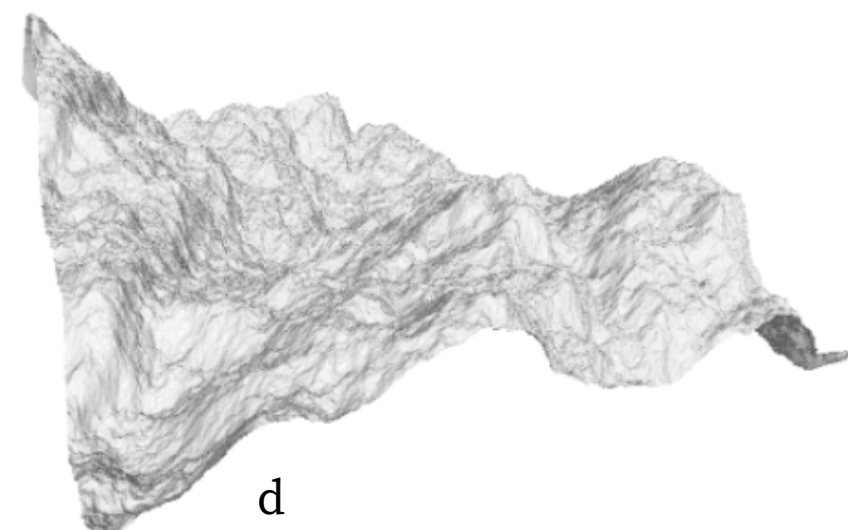
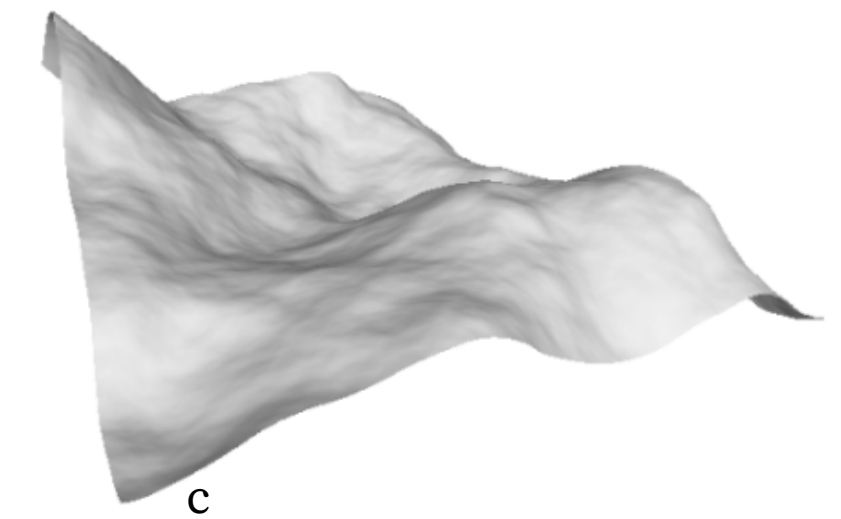
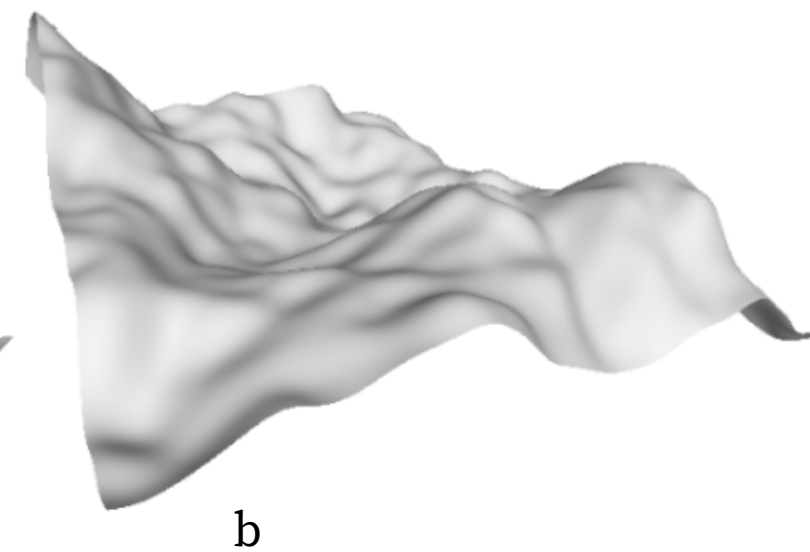
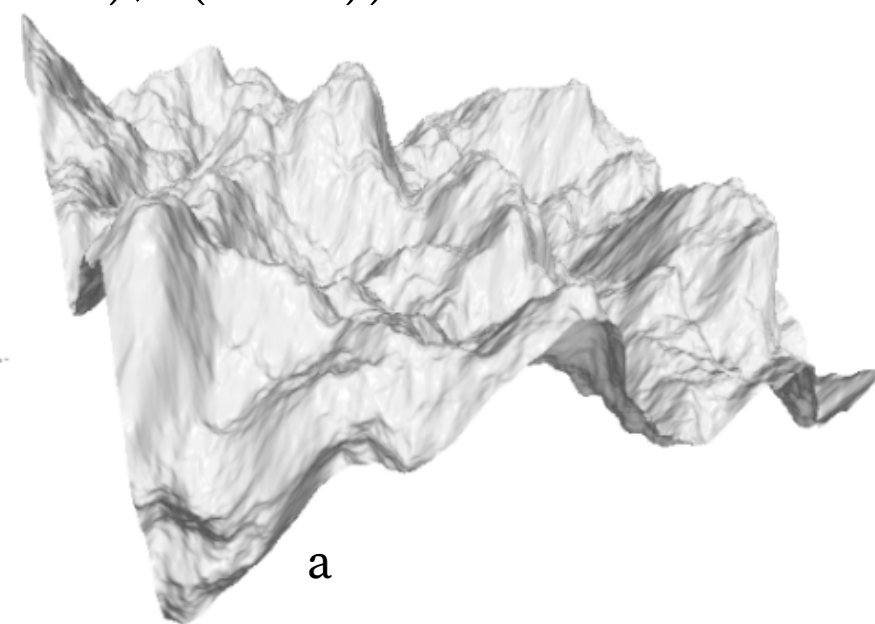
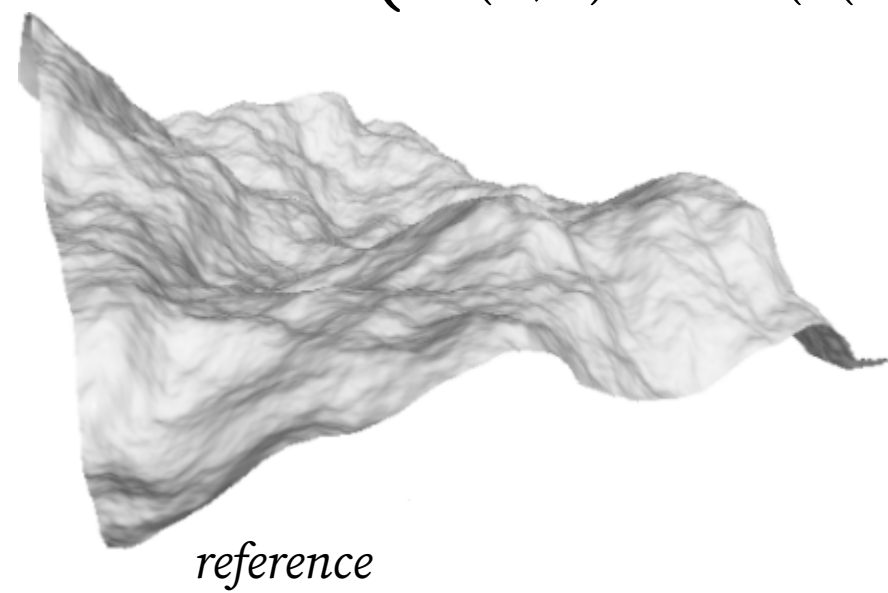
$$N = 9$$

$$\alpha = 0.4$$

$$h = 0.3$$

$$s = 1$$

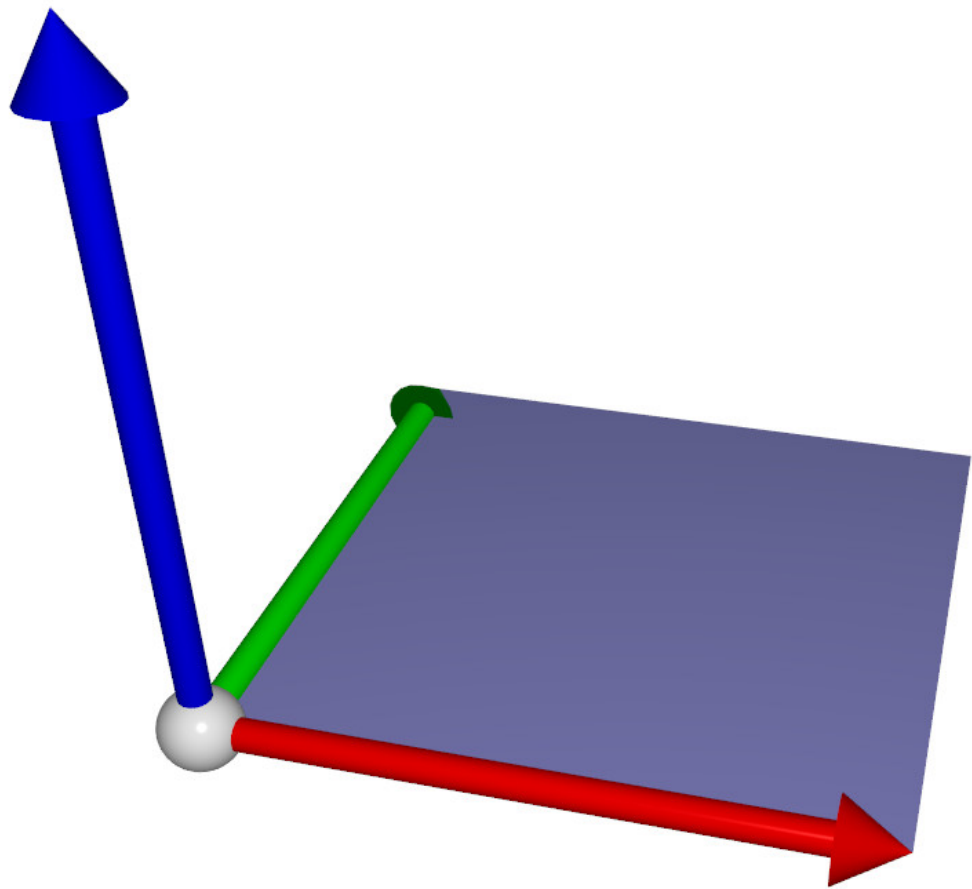
$$o = 0$$



Q. Which parameters correspond to (a,b,c,d,e,f) terrains?

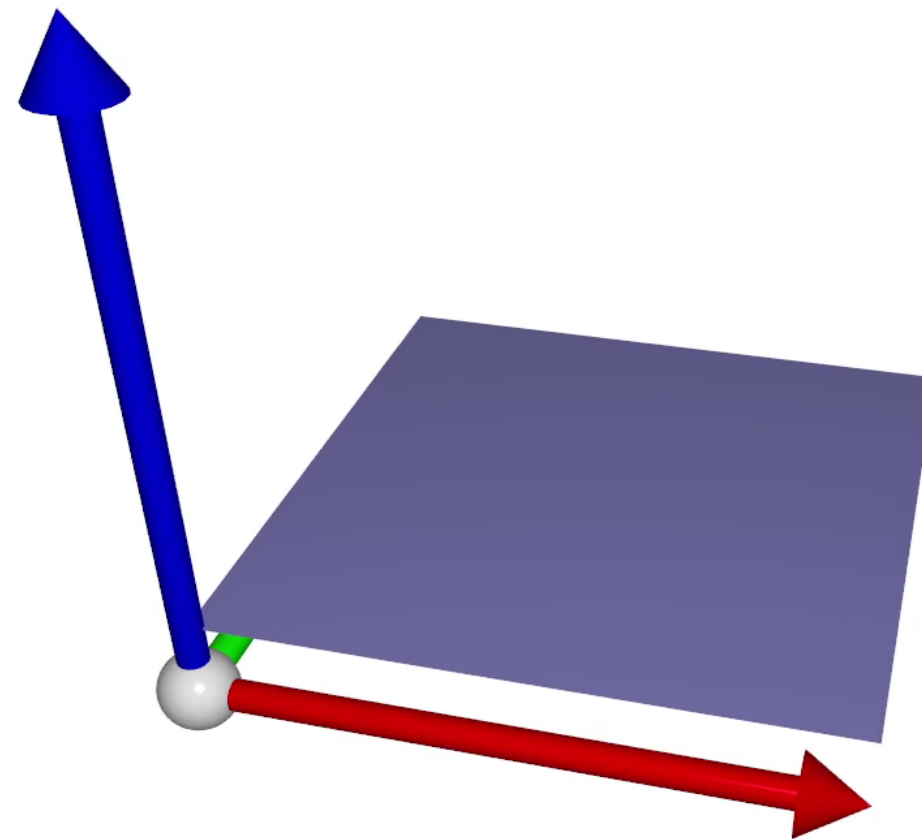
Perlin Noise - Animation

19/81



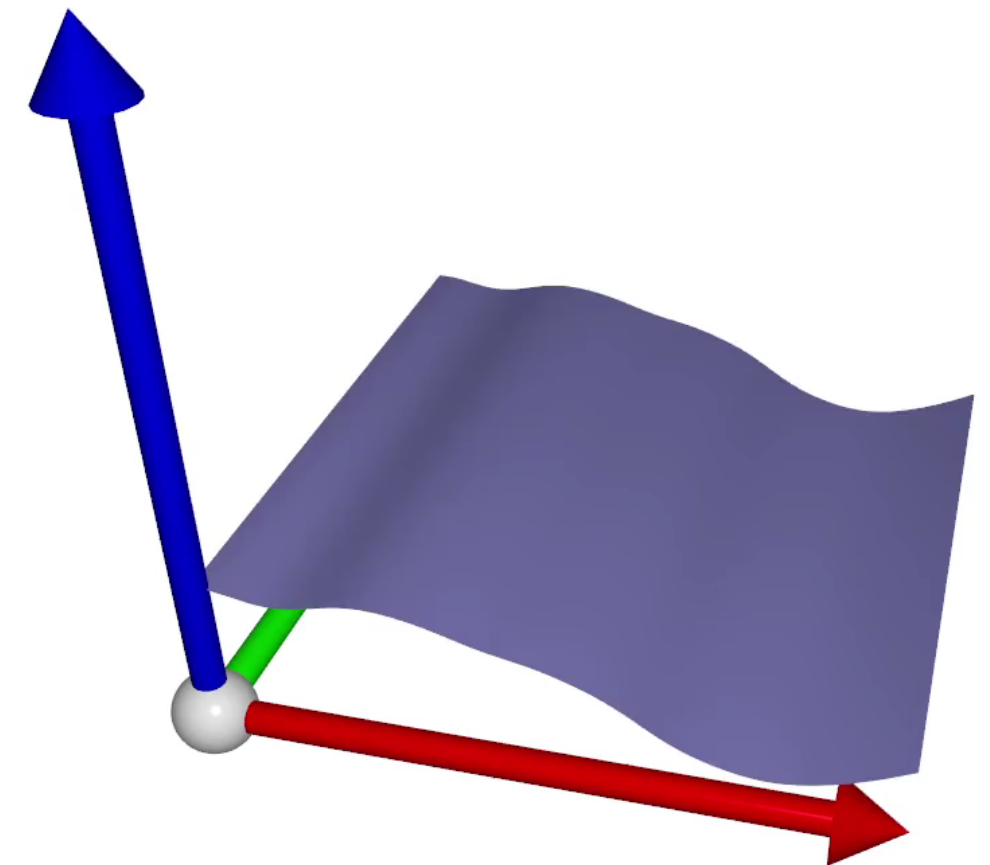
Reference surface function

$$(u, v) \in [0, 1]^2, S(u, v) = (u, v, 0)$$



Perlin noise can depends on time

$$S(u, v, t) = (u, v, P(t))$$



Depends on space and time

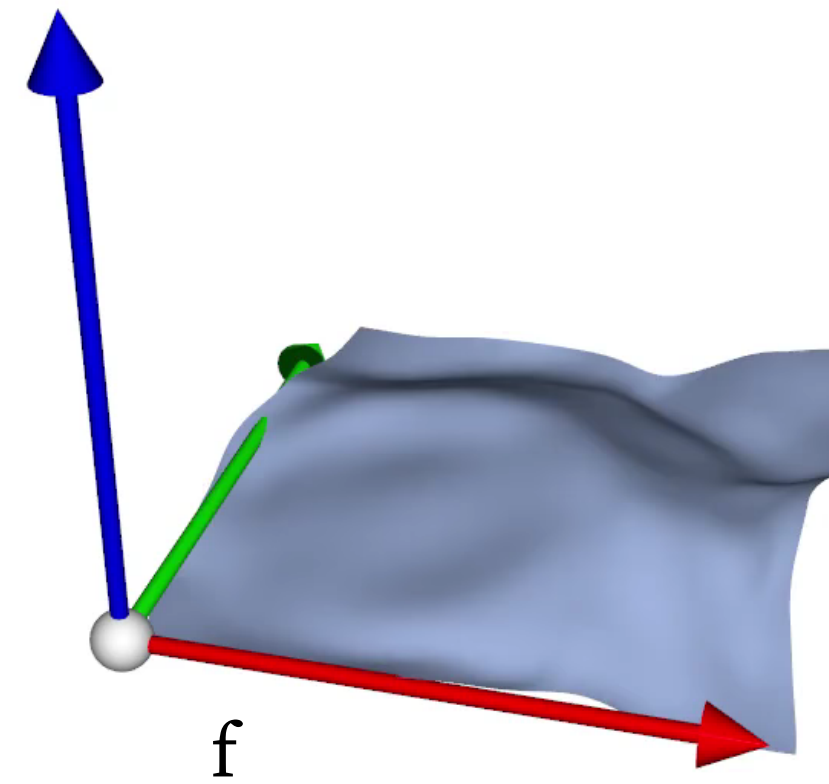
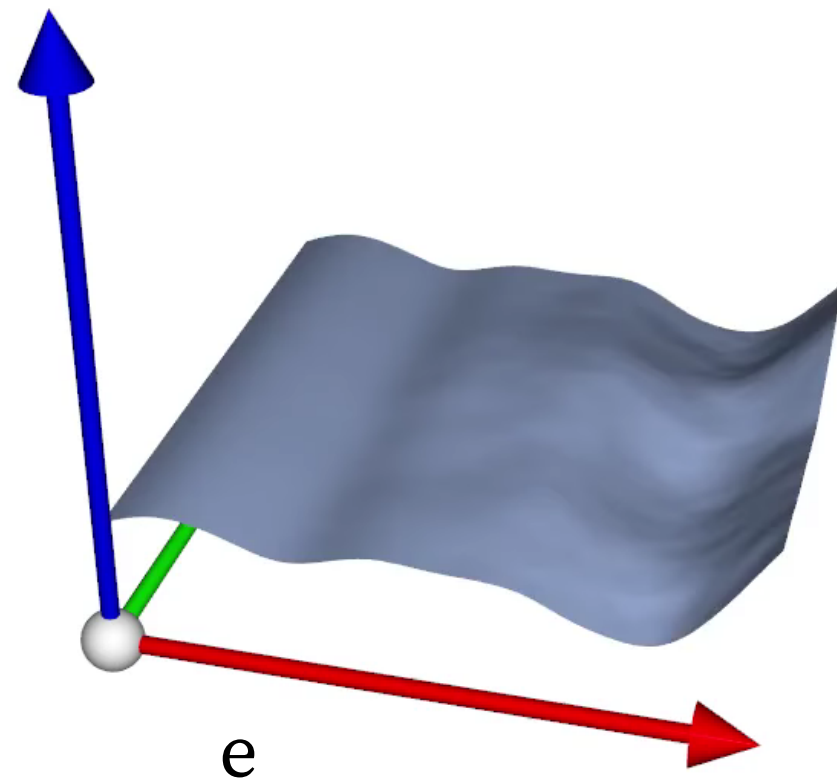
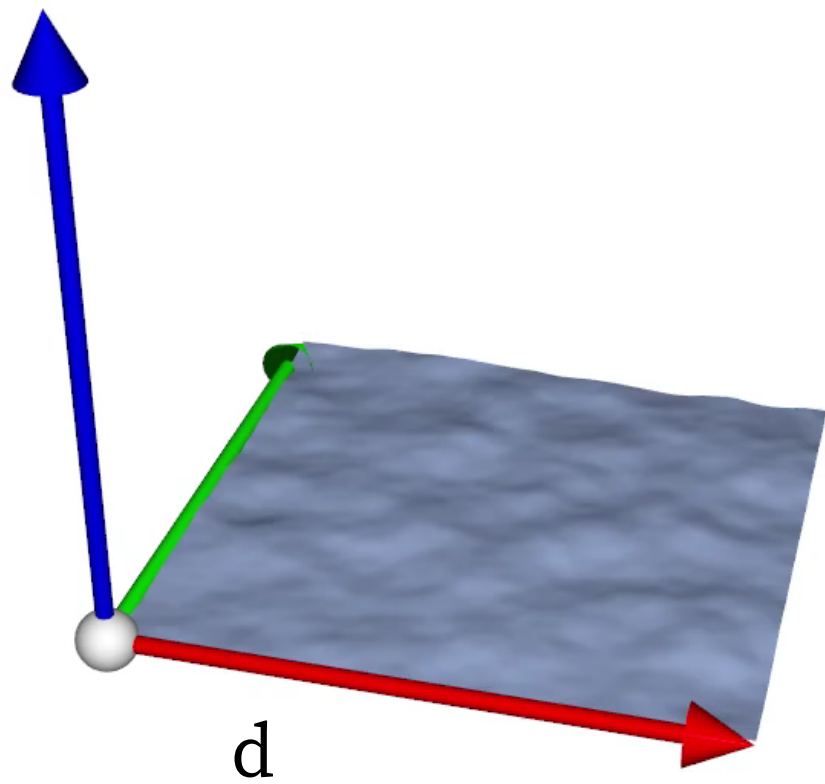
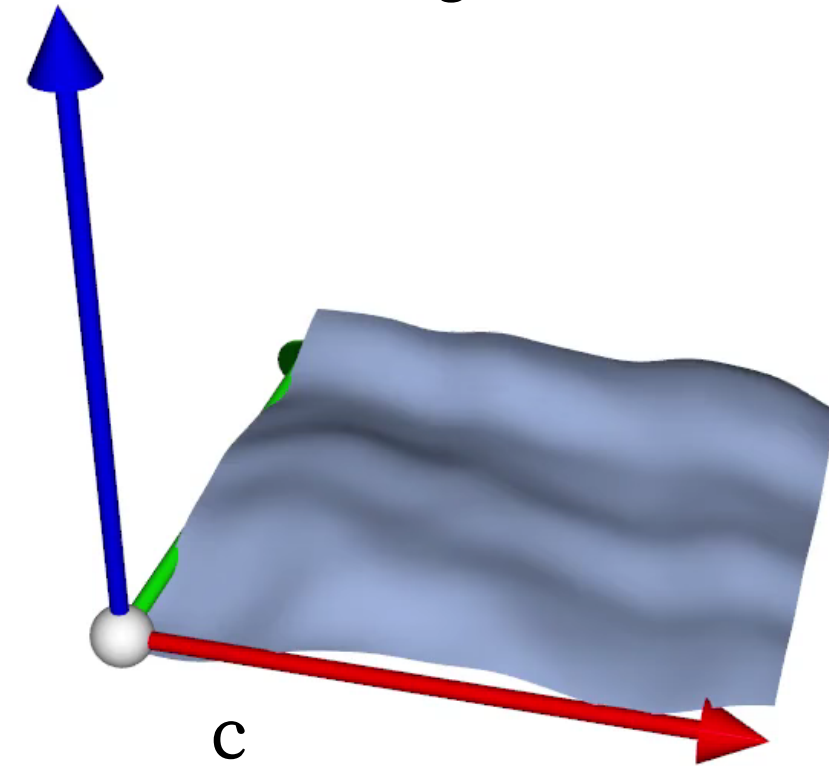
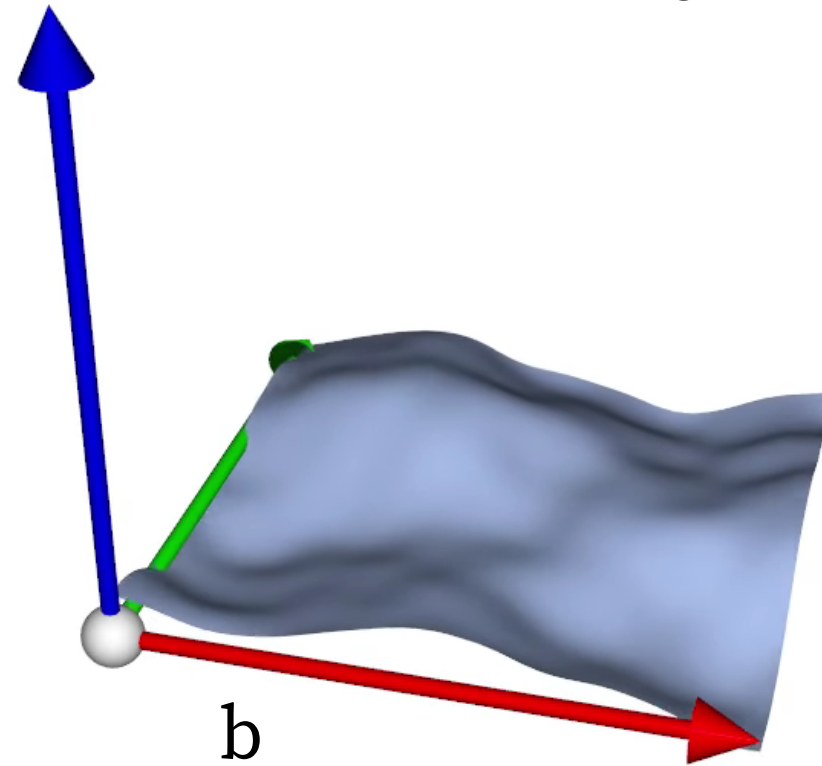
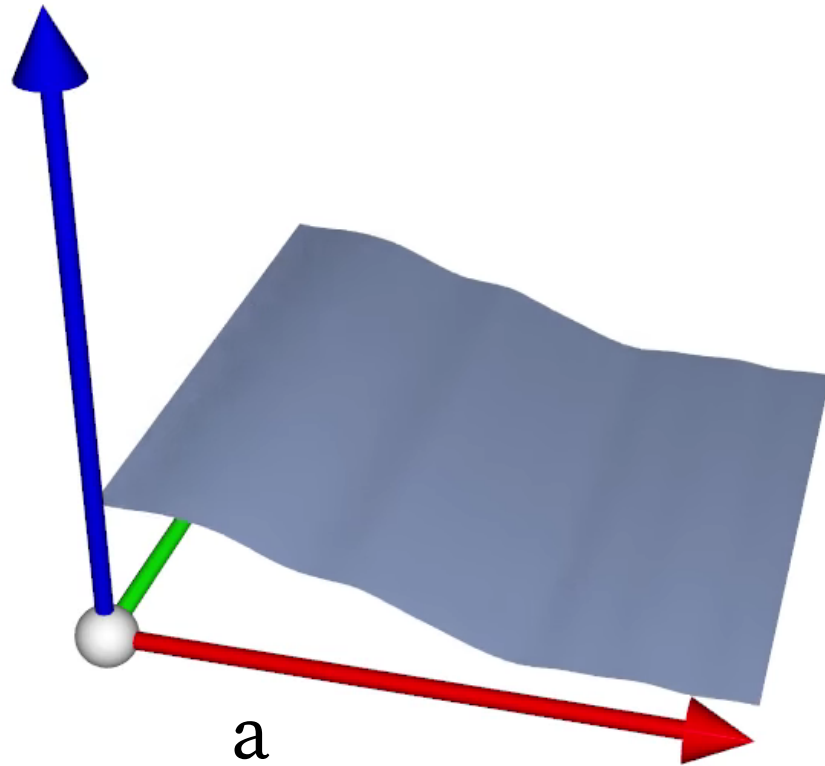
$$S(u, v, t) = (u, v, P(u, t))$$

Perlin Noise - Animation

20/81

Reference surface function: $(u, v) \in [0, 1]^2$, $f(u, v) = (u, v, 0)$

Q. How generate the following animations? $S(u, v, t) = \dots$



Affine Transformation

Affine/Linear Transformation

22/81

Most used deformations $p' = f(p)$

can be represented by a matrix $\rightarrow$ can be sent as uniform parameter to the shader.

- In standard 3D coordinates $p = (x, y, z)$

$$p' = L p + t$$

L : linear component (3×3 matrix)

t : translation

- In homogeneous coordinates $p = (x, y, z, 1)$

$$p' = A p$$

A : 4×4 matrix

$$A = \left(\begin{array}{ccc|c} L & t \\ \hline 0 & 1 \end{array} \right) = \left(\begin{array}{ccc|c} L_{xx} & L_{xy} & L_{xz} & t_x \\ L_{yx} & L_{yy} & L_{yz} & t_y \\ L_{zx} & L_{zy} & L_{zz} & t_z \\ \hline 0 & 0 & 0 & 1 \end{array} \right)$$

Properties:

- $\det(L)$: Change of volume when applying L to a shape

- $L^T L = 1 \Rightarrow$ Isometry

Affine/Linear Deformations

23/81

Translation

$$t(p) = (x + t_x, y + t_y, z + t_z)$$

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling

$$s(p) = (s_x x, s_y y, s_z z)$$

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation

Several possible representations
(see later)

Note: Isometry

Shearing

$$sh_{xy}(p) = (x + \lambda y, y, z)$$

$$sh_{xz}(p) = (x + \lambda z, y, z)$$

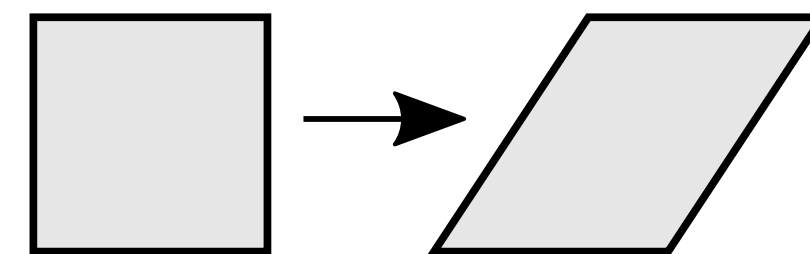
$$sh_{yx}(p) = (x, y + \lambda x, z)$$

...

$$Sh_{xy} = \begin{pmatrix} 1 & \lambda & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad Sh_{xz} = \begin{pmatrix} 1 & 0 & \lambda & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad Sh_{yx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \lambda & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \dots$$

Usually avoided in Graphics

Note: $\det(Sh) = 1 \rightarrow$ constant volume (but no isometry)



Rotations

24/81

3D rotations have 3 dof, No unique representation

Matrix

$$R = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix}$$

$$R^T R = I$$

$$\det(R) = 1$$

(+) Computationally convenient

(-) Non-explicit dof, redundancies

Euler Angles

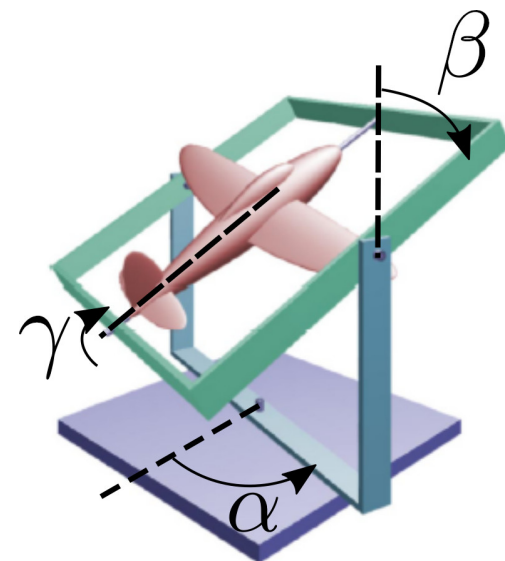
3 angles: (α, β, γ)

Composition of rotation around basic axes

Not unique (x-y-z, y-z-x, x-y-x', x-z-x', ...)

(+) Meaningfull parameters

(-) Gimbal-lock

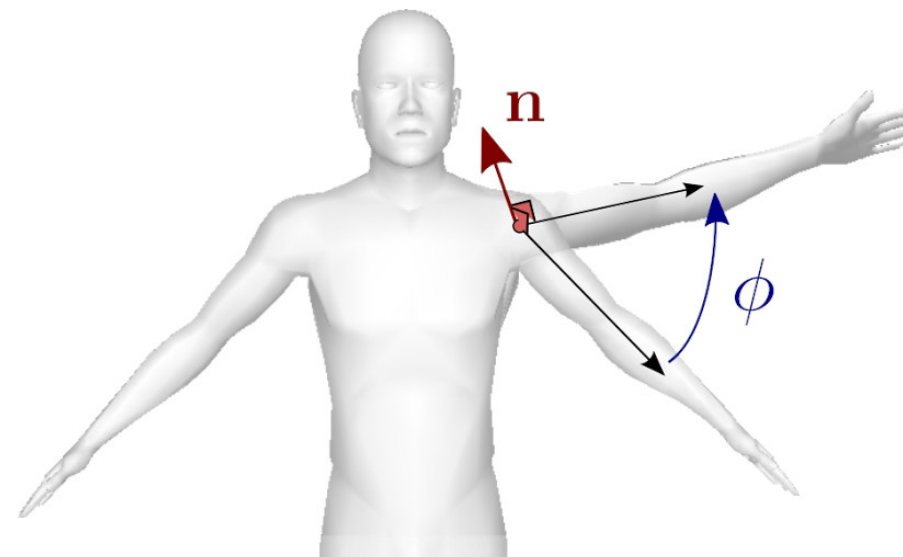


Axis angle

$(\mathbf{n}, \theta)$

(+) Meaningfull parameters

(-) No direct composition



Quaternion

$$q = (x, y, z, w) \\ = \left(\mathbf{n} \sin \left(\frac{\theta}{2} \right), \cos \left(\frac{\theta}{2} \right) \right)$$

(+) Composition and interpolation

(-) Less intuitive components

Cautions with transformations order

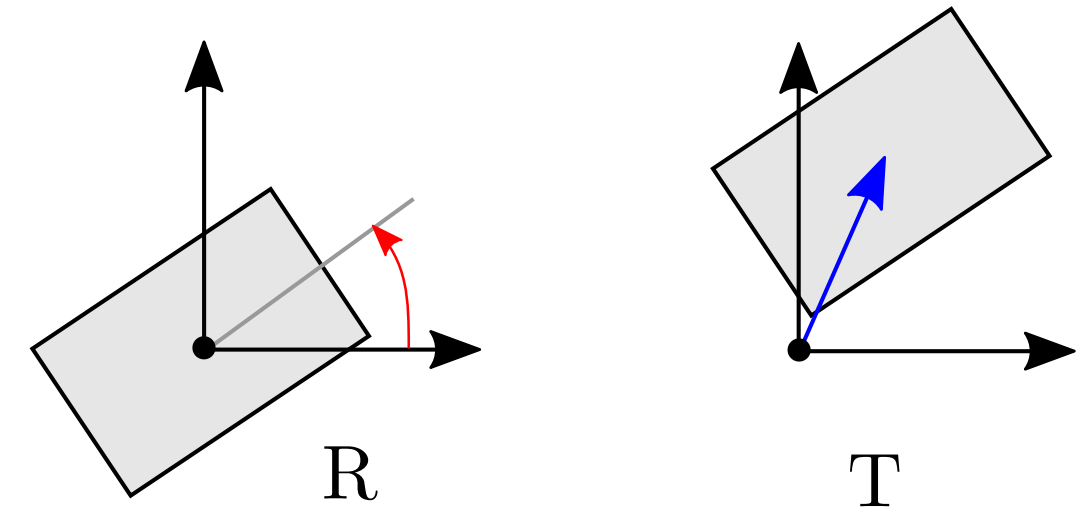
Take care, order of operation does matter !

Rotation r , Translation t : $r \circ t \neq t \circ r \Rightarrow M_1 = \mathbf{T} \mathbf{R} \neq \mathbf{R} \mathbf{T} = M_2$

Take care (2): transformation matrices applied to coordinates from right to left.

$$M_1 = \mathbf{T} \mathbf{R} = \left(\begin{array}{c|c} 1 & t \\ \hline 0 & 1 \end{array} \right) \left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array} \right) = \left(\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array} \right)$$

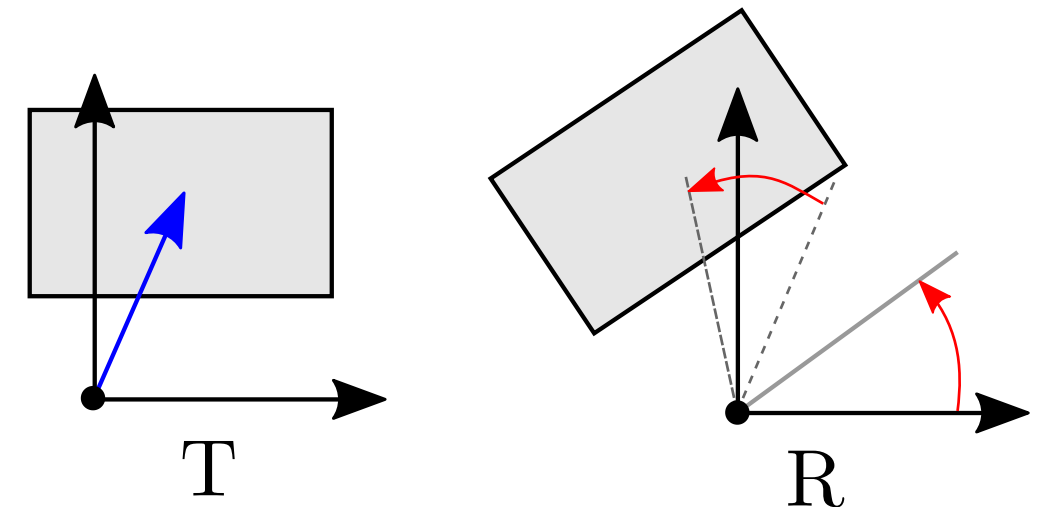
First rotates, then translates



$$M_2 = \mathbf{R} \mathbf{T} = \left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array} \right) \left(\begin{array}{c|c} 1 & t \\ \hline 0 & 1 \end{array} \right) = \left(\begin{array}{c|c} R & \mathbf{R} t \\ \hline 0 & 1 \end{array} \right)$$

First translates, then rotates

Rotation always happens around the origin.



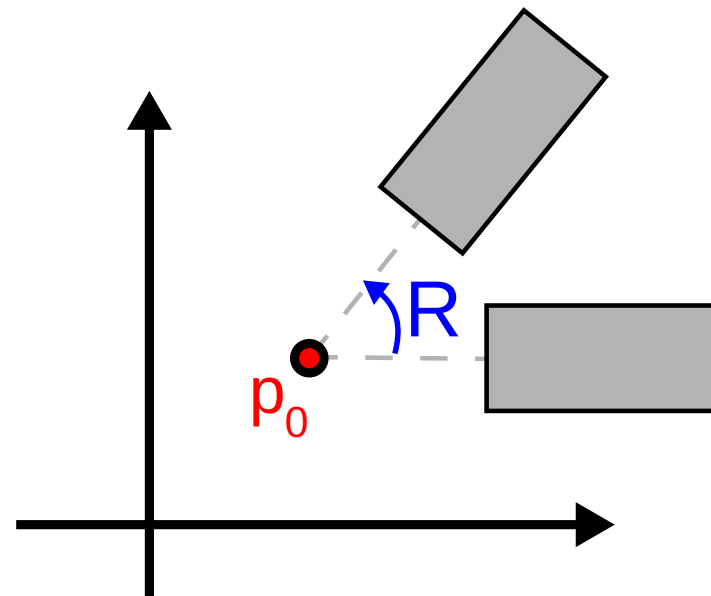
Beware: Some libraries (old OpenGL, Three.js) apply transformation from "left to right" using transposed matrix multiplication

Affine Transformation Exercises

26/81

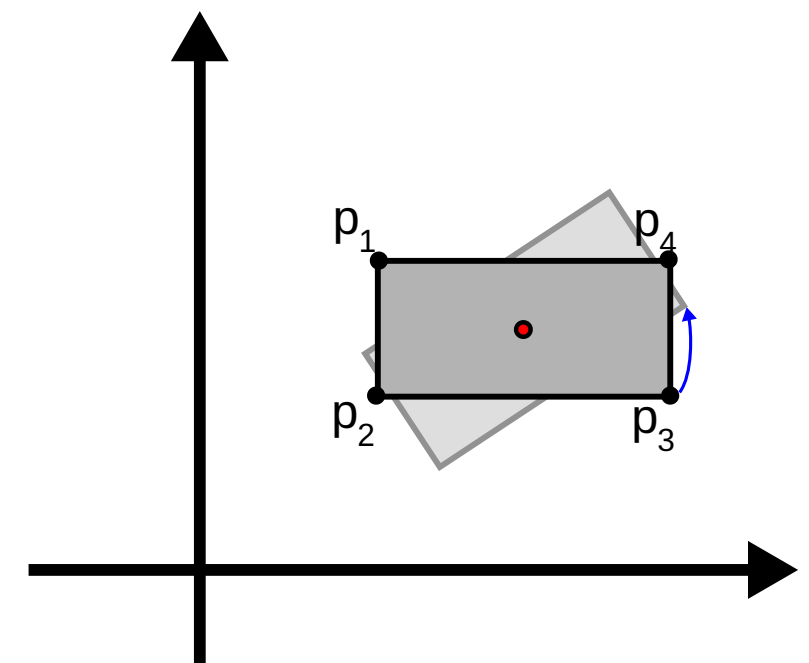
Q. Express the affine transform (as a 4×4 block matrix) corresponding to a rotation R applied around an arbitrary position p_0 in space.

> $M = \dots$



Q. Consider a shape defined by a triangular mesh with vertex positions $(p_i)_{i \in [1, N]}$.

> Express the affine transform allowing to rotate the shape around its barycenter.

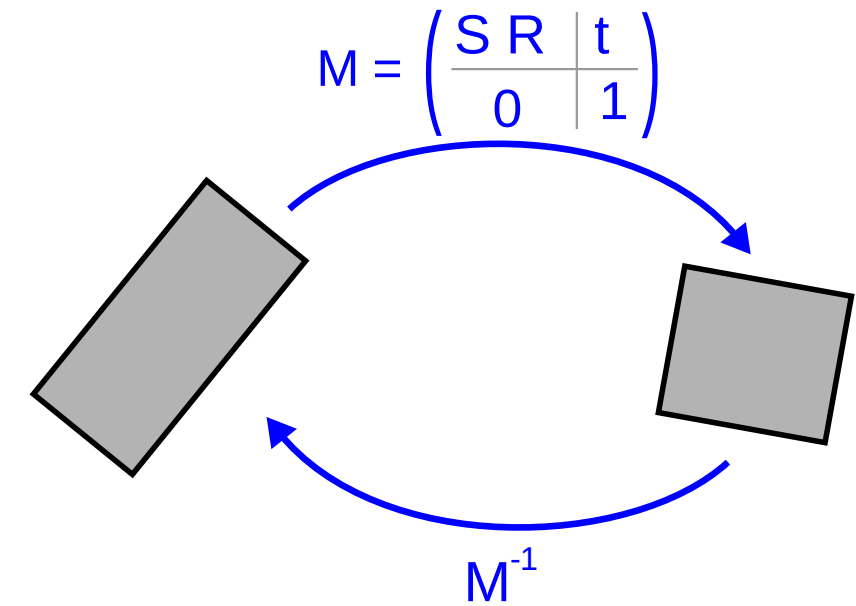


Affine Transformation Exercises

27/81

Q. Consider the affine transform M parameterized by a scaling s , a rotation R , and a translation t .

> Express the inverse matrix M^{-1} with respect to s , R and t

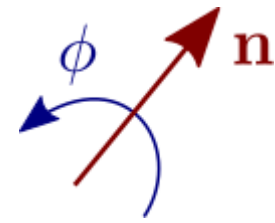


Rotation: Focus Axis Angle

28/81

Any 3D rotation can be represented by

- A unit axis n
- An angle θ

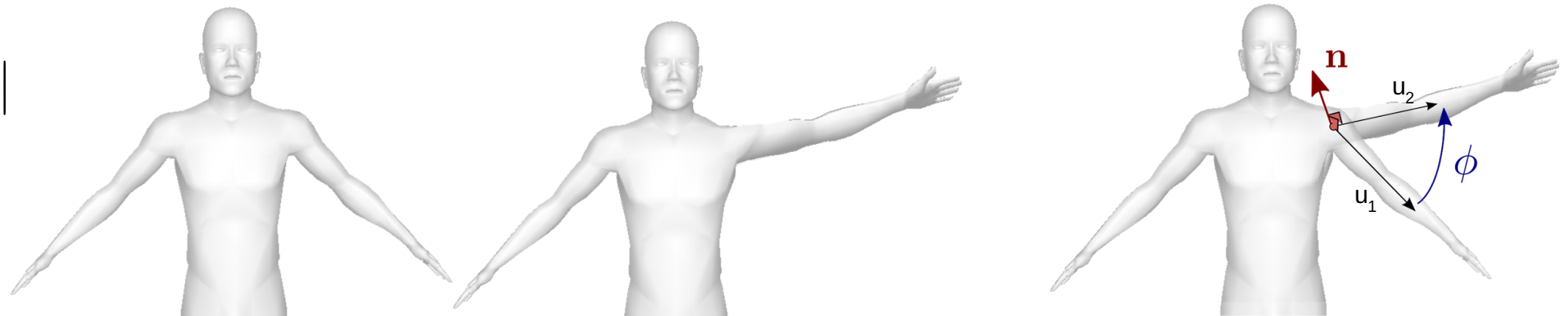


Pro.

- Concise representation: 3 DOF
- Meaningfull parameters
- Describes well the rotation between two vectors

Ex. Rotation between u_1 and u_2

- Axis of rotation $n = (u_1 \times u_2) / \|u_1 \times u_2\|$
- Angle of rotation $\theta = \text{acos}(u_1 \cdot u_2)$
- Twist around u_2 can be arbitrarily chosen



Rotation: Focus Axis Angle - Rodrigues

29/81

Applying a rotation (n, θ) to a vector v

$$v = v_{\parallel} + v_{\perp}$$

$$v' = v'_{\parallel} + v'_{\perp}$$

$$\Rightarrow v' = v_{\parallel} + (\cos(\theta) v_{\perp} + \sin(\theta) n \times v_{\perp})$$

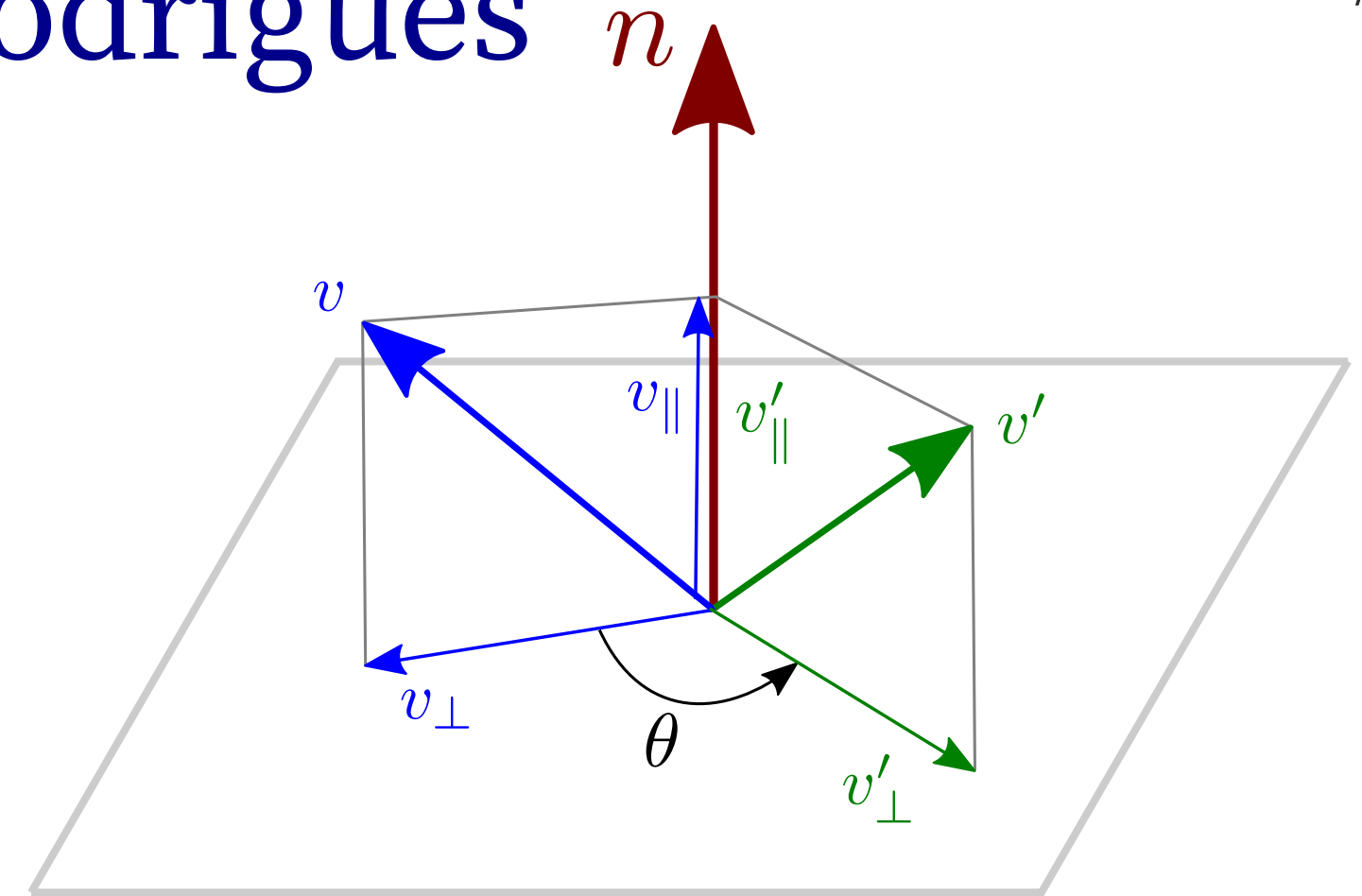
$$v_{\parallel} = (v \cdot n) n$$

$$v_{\perp} = v - (v \cdot n) n$$

$$v' = (v \cdot n) n + \cos(\theta)(v - (v \cdot n) n) + \sin(\theta) n \times (v - (v \cdot n) n)$$

$$\Rightarrow v' = \cos(\theta) v + \sin(\theta) n \times v + (1 - \cos(\theta)) (v \cdot n) n$$

Rodrigues' rotation Formula



Rotation: Focus Axis Angle as Matrix

30/81

$$v' = \cos(\theta) v + \sin(\theta) n \times v + (1 - \cos(\theta)) (v \cdot n) n = R(n, \theta) v$$

$$v' = \cos(\theta) v + \sin(\theta) \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \times v + (1 - \cos(\theta)) \begin{pmatrix} n_x & n_y & n_z \end{pmatrix} v \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

$$v' = \cos(\theta) v + \sin(\theta) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_K v + (1 - \cos(\theta)) \underbrace{\begin{pmatrix} n_x^2 & n_x n_y & n_x n_z \\ n_x n_y & n_y^2 & n_y n_z \\ n_x n_z & n_y n_z & n_z^2 \end{pmatrix}}_{K^2} v$$

$$v' = \begin{pmatrix} \cos(\theta) + n_x^2(1 - \cos(\theta)) & n_x n_y(1 - \cos(\theta)) - n_z \sin(\theta) & n_x n_z(1 - \cos(\theta)) + n_y \sin(\theta) \\ n_x n_y(1 - \cos(\theta)) + n_z \sin(\theta) & \cos(\theta) + n_y^2(1 - \cos(\theta)) & n_y n_z(1 - \cos(\theta)) - n_x \sin(\theta) \\ n_x n_z(1 - \cos(\theta)) - n_y \sin(\theta) & n_y n_z(1 - \cos(\theta)) + n_x \sin(\theta) & \cos(\theta) + n_z^2(1 - \cos(\theta)) \end{pmatrix} v$$

Rotation: Focus Axis Angle - Summary

31/81

Given a rotation (n, θ) - Corresponding rotation matrix

$$R(n, \theta) = I + \sin(\theta) K + (1 - \cos(\theta)) K^2$$

$$R(n, \theta) = \begin{pmatrix} \cos(\theta) + n_x^2(1 - \cos(\theta)) & n_x n_y(1 - \cos(\theta)) - n_z \sin(\theta) & n_x n_z(1 - \cos(\theta)) + n_y \sin(\theta) \\ n_x n_y(1 - \cos(\theta)) + n_z \sin(\theta) & \cos(\theta) + n_y^2(1 - \cos(\theta)) & n_y n_z(1 - \cos(\theta)) - n_x \sin(\theta) \\ n_x n_z(1 - \cos(\theta)) - n_y \sin(\theta) & n_y n_z(1 - \cos(\theta)) + n_x \sin(\theta) & \cos(\theta) + n_z^2(1 - \cos(\theta)) \end{pmatrix}$$

Pro

- Concise and general representation
- Expressive parameters in 3D space (axe, angles)
- Efficient rotation and correspondance with matrix

Cons

- No simple composition expression between two rotations.
- No direct interpolation

(well handled by quaternion representation)

Rotation: Focus on Quaternions

32/81

Quaternions: generalization of complex numbers.

$$q = x \mathbf{i} + y \mathbf{j} + z \mathbf{k} + w \quad w \text{ real part, } (x, y, z) \text{ imaginary (or pure quaternion) part.}$$

We write in short $q = (x, y, z, w)$

(don't confound with 4D vectors in homogeneous coordinates)

Properties of imaginary basis vectors

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$$

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k}$$

$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i}$$

$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j}$$

$$\mathbf{ijk} = -1$$

- Provides the algebraic properties

Basics operations on quaternions

33/81

- Conjugated quaternion $q^* = (-x, -y, -z, w)$

- Quaternion norm $\|q\| = \sqrt{q q^*} = \sqrt{x^2 + y^2 + z^2 + w^2}$. Unit quaternion satisfies $\|q\| = 1$.

- Quaternion product

$$q_1 q_2 = (x_1 \mathbf{i} + y_1 \mathbf{j} + z_1 \mathbf{k} + w_1) (x_2 \mathbf{i} + y_2 \mathbf{j} + z_2 \mathbf{k} + w_2) = \dots$$

$$q_1 q_2 = \begin{pmatrix} x_1 w_2 + w_1 x_2 + y_1 z_2 - z_1 y_2 \\ y_1 w_2 + w_1 y_2 + z_1 x_2 - x_1 z_2 \\ z_1 w_2 + w_1 z_2 + x_1 y_2 - y_1 x_2 \\ w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \end{pmatrix}$$

Note: Quaternion product is

- associative: $(q_1 q_2) q_3 = q_1 (q_2 q_3) = q_1 q_2 q_3$

- **non-commutative**: $q_1 q_2 \neq q_2 q_1$

Sometimes interesting to separate *real part* w from *pure quaternion part* $s = (x, y, z)$.

- Shorthand vector form $q = (s, w)$

- Quaternion product in vector form $q_1 q_2 = (s_1 w_2 + s_2 w_1 + s_1 \times s_2, w_1 w_2 - s_1 \cdot s_2)$

Relation between quaternion and rotation

34/81

Consider

- a quaternion $q = (s, w)$ of unit norm $\|q\| = 1$.
- a vector $v = (v_x, v_y, v_z)$ assimilated to the pure quaternion $q_v = (v, 0) = (v_x, v_y, v_z, 0)$.

Then

- $q_{v'} = \mathcal{R}_q(v) = q q_v q^*$ is a pure quaternion $q_{v'} = (v'_x, v'_y, v'_z, 0)$
- And $v' = (v'_x, v'_y, v'_z)$ is the rotation of vector v around the axis $n = s/\|s\|$, with angle $2 \arccos(w)$.

Demonstration

$$\mathcal{R}_q(v) = (s, w) (v, 0) (-s, w) = \dots = ((w^2 - s^2) v + 2(s \cdot v) s + 2w (s \times v), 0)$$

As $\|q\| = 1$, we can write $q = (s, w) = (n \sin(\phi), \cos(\phi))$, where $\|n\| = 1$

$$\text{Then } \mathcal{R}_q(v) = \underbrace{(\cos^2(\phi) - \sin^2(\phi))}_{\cos(2\phi)} v + \underbrace{2 \sin^2(\phi)}_{1 - \cos(2\phi)} (n \cdot v) n + \underbrace{2 \cos(\phi) \sin(\phi)}_{\sin(2\phi)} n \times v, 0)$$

$\Rightarrow$ Rodrigues formula for axis n and angle 2ϕ .

The unit quaternion $q = (n \sin(\theta/2), \cos(\theta/2))$ represents the rotation of angle θ around the axis n .

Composition of rotations

35/81

Consider two rotation (R_1, R_2) associated to their unit quaternions (q_1, q_2) .

The product $q_1 q_2$ represents the composition $R_1 \circ R_2$.

Demonstration

We show that $\mathcal{R}_{q_1 q_2}(v) = \mathcal{R}_{q_1} \circ \mathcal{R}_{q_2}(v)$.

$$\mathcal{R}_{q_1 q_2}(v) = (q_1 q_2) v (q_1 q_2)^*$$

$$\mathcal{R}_{q_1 q_2}(v) = (q_1 q_2) v (q_2^* q_1^*), \text{ as } (q_1 q_2)^* = q_2^* q_1^*$$

$$\mathcal{R}_{q_1 q_2}(v) = q_1 (q_2 v q_2^*) q_1^*$$

$$\mathcal{R}_{q_1 q_2}(v) = q_1 \mathcal{R}_{q_2}(v) q_1^* = \mathcal{R}_{q_1} \circ \mathcal{R}_{q_2}(v)$$

Correspondance quaternion to rotation matrix

36/81

The unit quaternion $q = (x, y, z, w)$ represents the rotation given by the matrix

$$R = \begin{pmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{pmatrix}$$

Demonstration

$$v' = \mathcal{R}_q(v) = q q_v q^\star = ((w^2 - s^2) v + 2(s \cdot v) s + 2w (s \times v), 0) \quad \text{with } s = (x, y, z)$$

$$v' = (w^2 - x^2 - y^2 - z^2)v + 2 \begin{pmatrix} x & y & z \end{pmatrix} v \begin{pmatrix} x & y & z \end{pmatrix}^T + 2w \begin{pmatrix} x & y & z \end{pmatrix} \times v$$

$$v' = \left((w^2 - x^2 - y^2 - z^2) \mathbf{I} + 2 \begin{pmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{pmatrix} + 2w \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \right) v$$

$$v' = R v, \text{ with } x^2 + y^2 + z^2 + w^2 = 1$$

Summary - Correspondance quaternion / matrix-vector

37/81

Representation and Operations

	3D space	Quaternion space
<i>Vector</i>	$v = (v_x, v_y, v_z)$	$q_v = (v, 0) = (v_x, v_y, v_z, 0)$
<i>Rotation</i>	R (3×3 matrix)	$q = (x, y, z, w), \ q\ = 1$
<i>Apply rotation to vector</i>	Rv	$q q_v q^*$
<i>Rotation composition</i>	$R_1 R_2$	$q_1 q_2$

Rotation to Quaternion

Rotation of axis n and angle $\theta \Rightarrow q = (n \sin(\theta/2), \cos(\theta/2))$

Quaternion to Rotation

Unit quaternion $q = (x, y, z, w) \Rightarrow R = \begin{pmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{pmatrix}$

Interpolating rotation

Do not use componentwise-interpolation on rotation matrix
 $\Rightarrow$ interpolate in **quaternion** space

Can use either:

- **SLERP** - Spherical Linear Interpolation

$$q(t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)} q_1 + \frac{\sin(t\Omega)}{\sin(\Omega)} q_2, \quad \text{with } \cos(\Omega) = q_1 \cdot q_2$$

Between two unit quaternions q_1, q_2

- **LERP** - Linear Interpolation

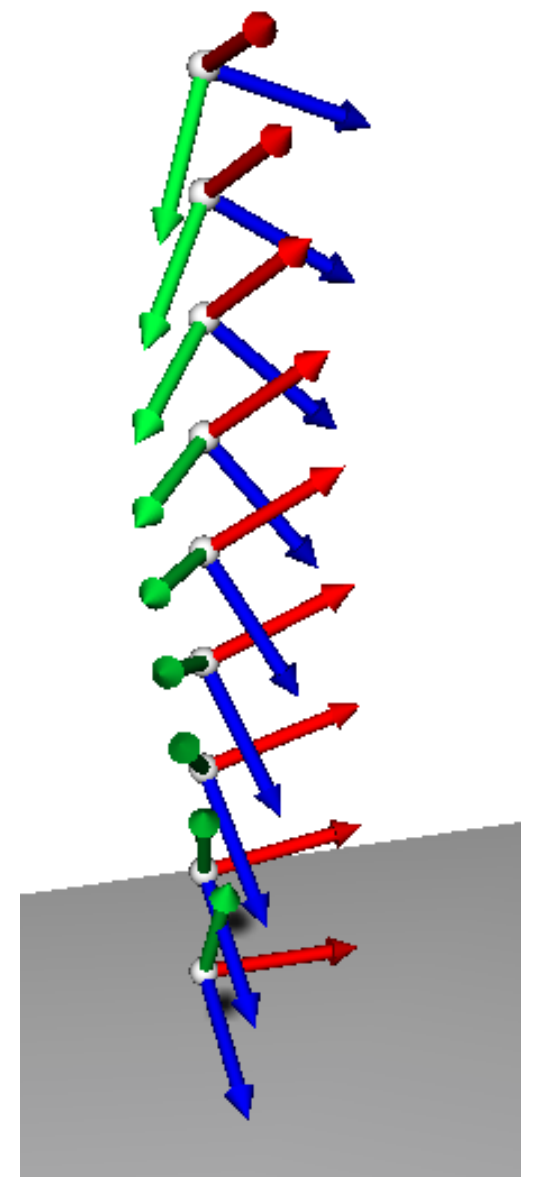
$$q(t) = \frac{\sum_j \alpha_j q_j}{\|\sum_j \alpha_j q_j\|}$$

When blending multiple quaternions q_j with weights α_j

Rem.

When interpolating b/w rotations and positions:

Use quaternion with rotation, componentwise-interpolation on position



Care with quaternion negation

39/81

$+q$ and $-q$ correspond to the same rotation ($n \rightarrow -n, \theta \rightarrow 2\pi - \theta$)

Warning $-q$ **does not** corresponds to the rotation matrix $-R$.

But to a **different path** when interpolated in the 4D quaternion space.

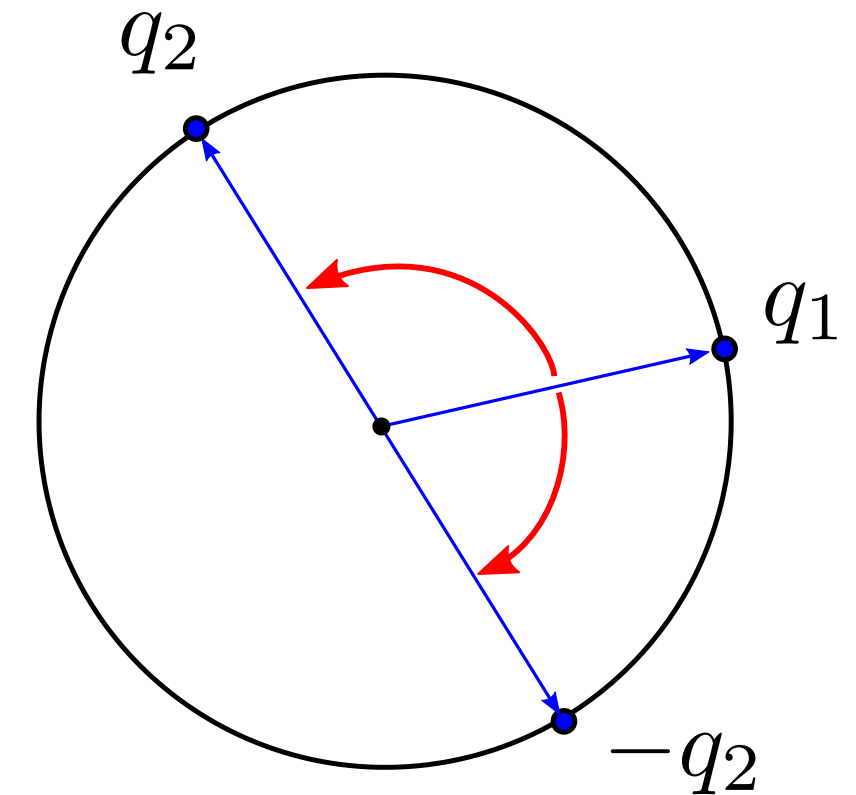
$\Rightarrow$ path $q_1 \rightarrow -q_2$ is shorter than $q_1 \rightarrow q_2$ when $q_1 \cdot q_2 < 0$.

In practice we check for the shorter path before applying SLERP.

Algorithm

```
if( dot(q1,q2)<0 )  
    q2 = -q2
```

```
q(t) = SLERP(q1,q2,t)
```



Animating Virtual Characters

Skin deformation

Skeleton based deformation - Skinning

42/81

Objective: Deform articulated character

Idea: Use skeleton to control limbs

Articulations as rotations

Animation Skeleton

Set of frames T_i : position, orientation

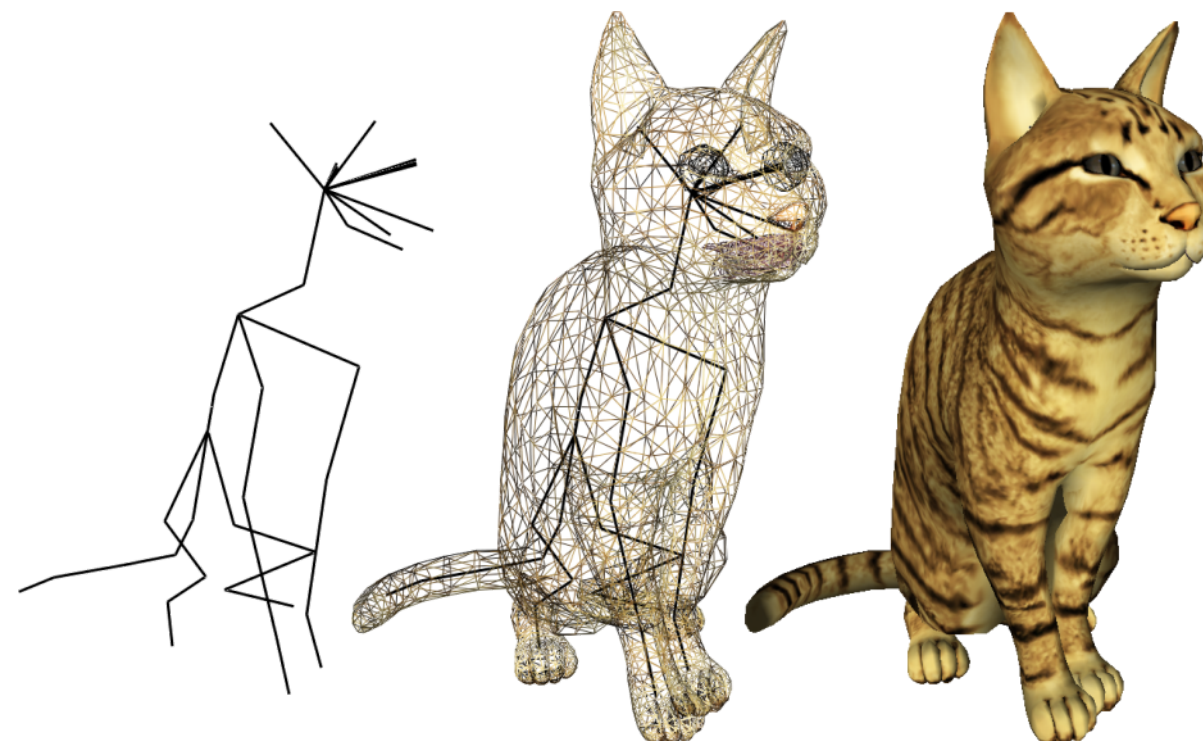
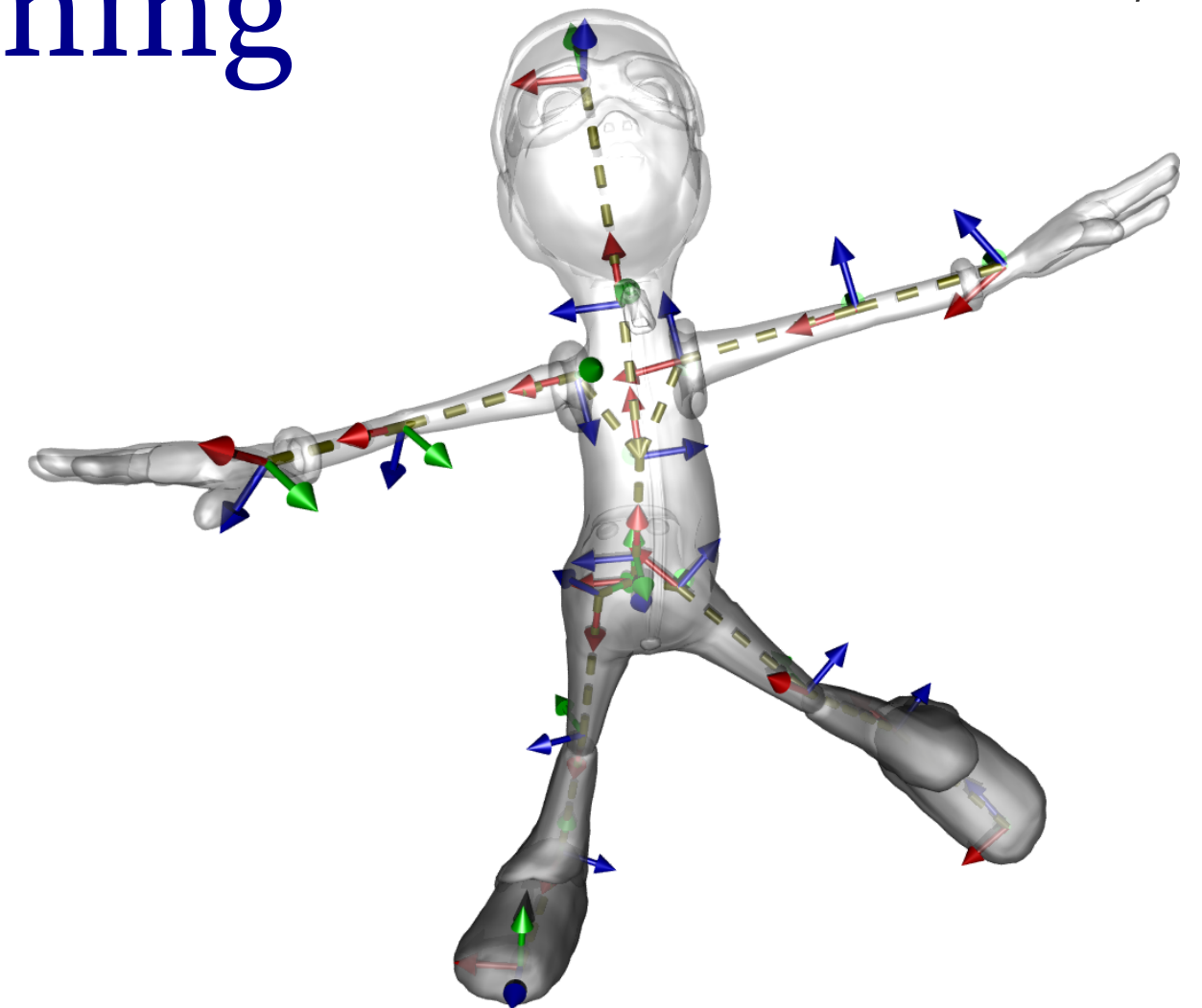
Describe non-rigid parts of the character (dof)

Terminology

Joint = A frame T_i

Bone = Segment between two joints

Note: Animation skeleton $\neq$ Anatomical skeleton



Rigid Skinning

43/81

Attach rigidly subset of vertices to specific bones, described by its root joint/frame.
Vertices are following rigid deformation of their associated frame.

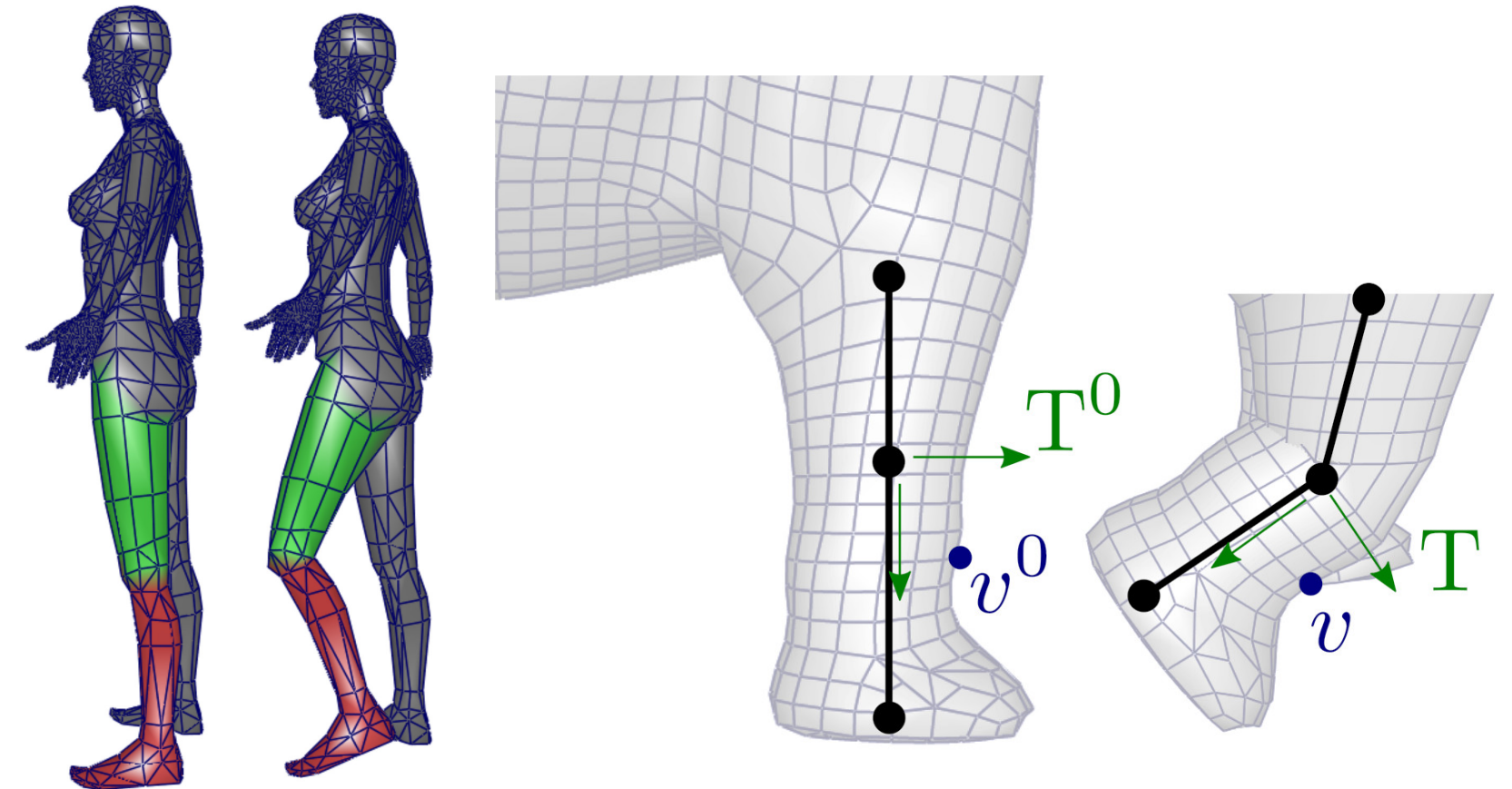
Deformation formulation

Consider, at rest pose/initial state

- A frame T^0
- A vertex v^0 attached to this frame

After deformation

- New frame T
- New vertex position v



Question: What are the new coordinates v w/r T , T^0 , v^0 ?

Rigid Skinning

44/81

Attach rigidly subset of vertices to specific bones, described by its root joint/frame.
Vertices are following rigid deformation of their associated frame.

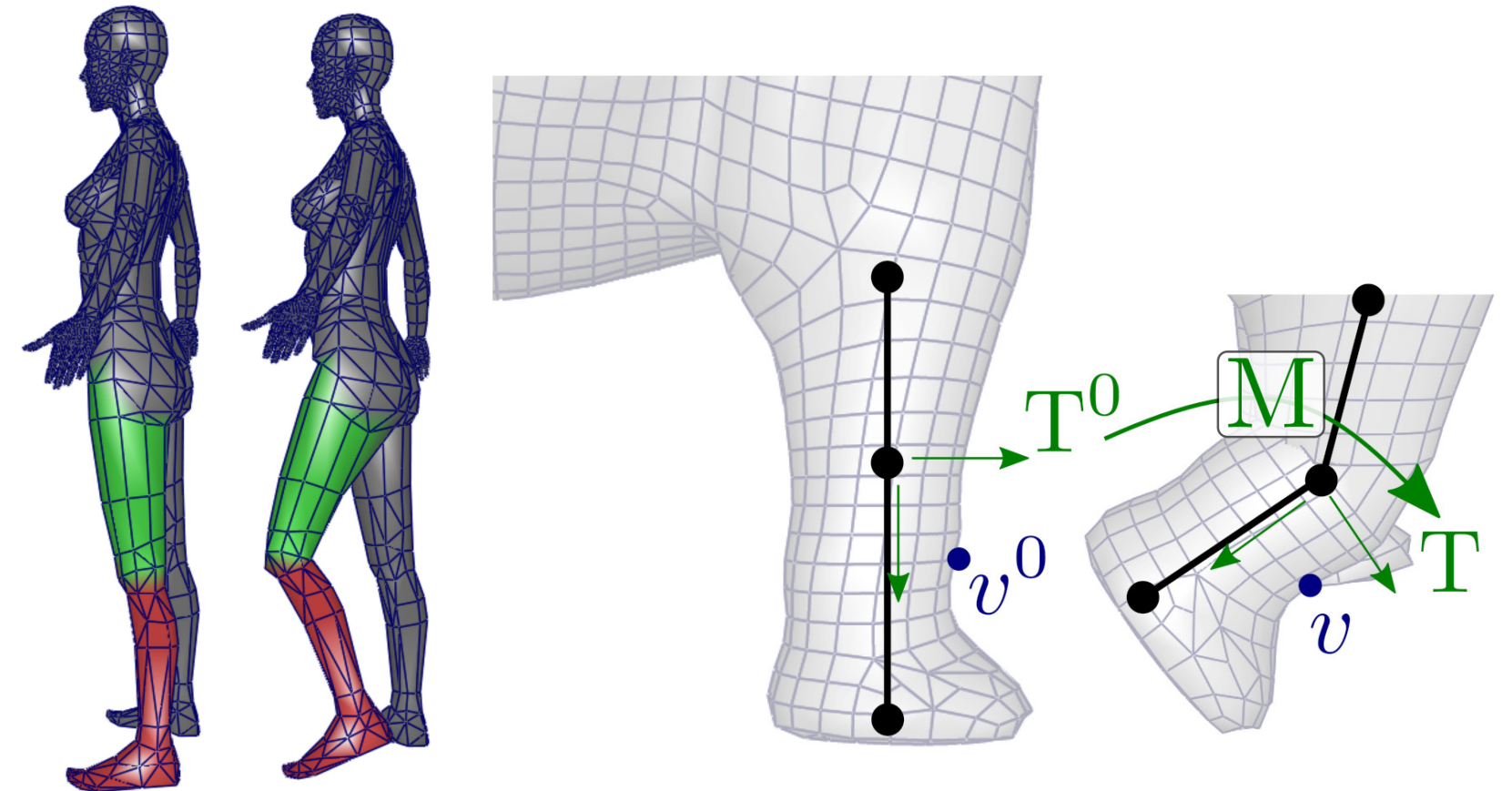
Deformation formulation

Consider, at rest pose/initial state

- A frame T^0
- A vertex v^0 attached to this frame

After deformation

- New frame T
- New vertex position v



Question: What are the new coordinates v w/r T , T^0 , v^0 ?

v^0 and v have similar local coordinates w/r to T^0 and T

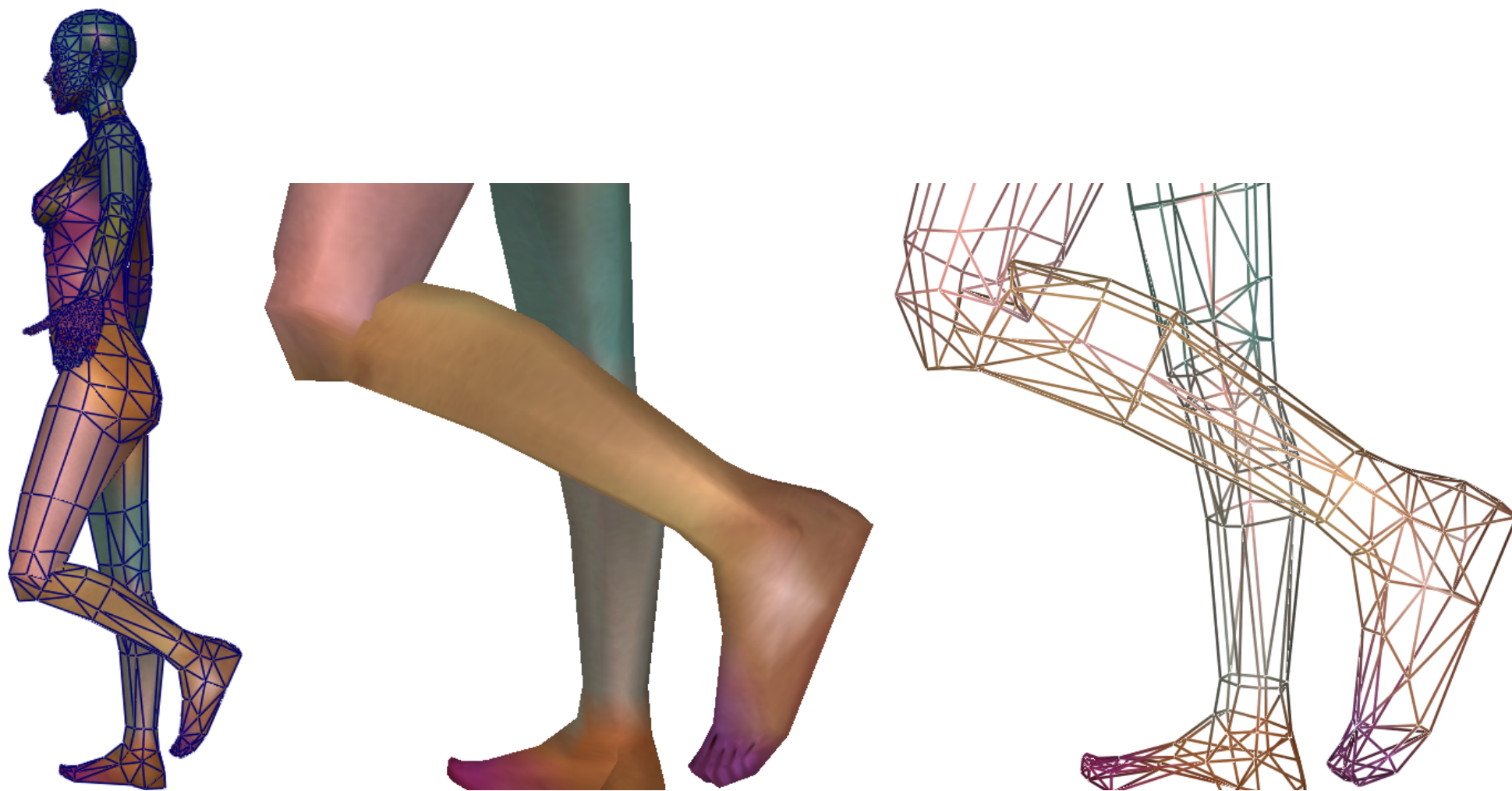
$$\Rightarrow T^{-1}v = (T^0)^{-1}v_0$$

$$\Rightarrow v = T (T^0)^{-1}v_0 = M v_0$$

Rigid Skinning

45/81

- (+) Skeleton is easy to build
- (+) Skeleton interaction is intuitive to model rigid articulation
- (-) Discontinuities/Inter-penetrations



Idea of smooth skinning: Blend discontinuous transformation around articulation

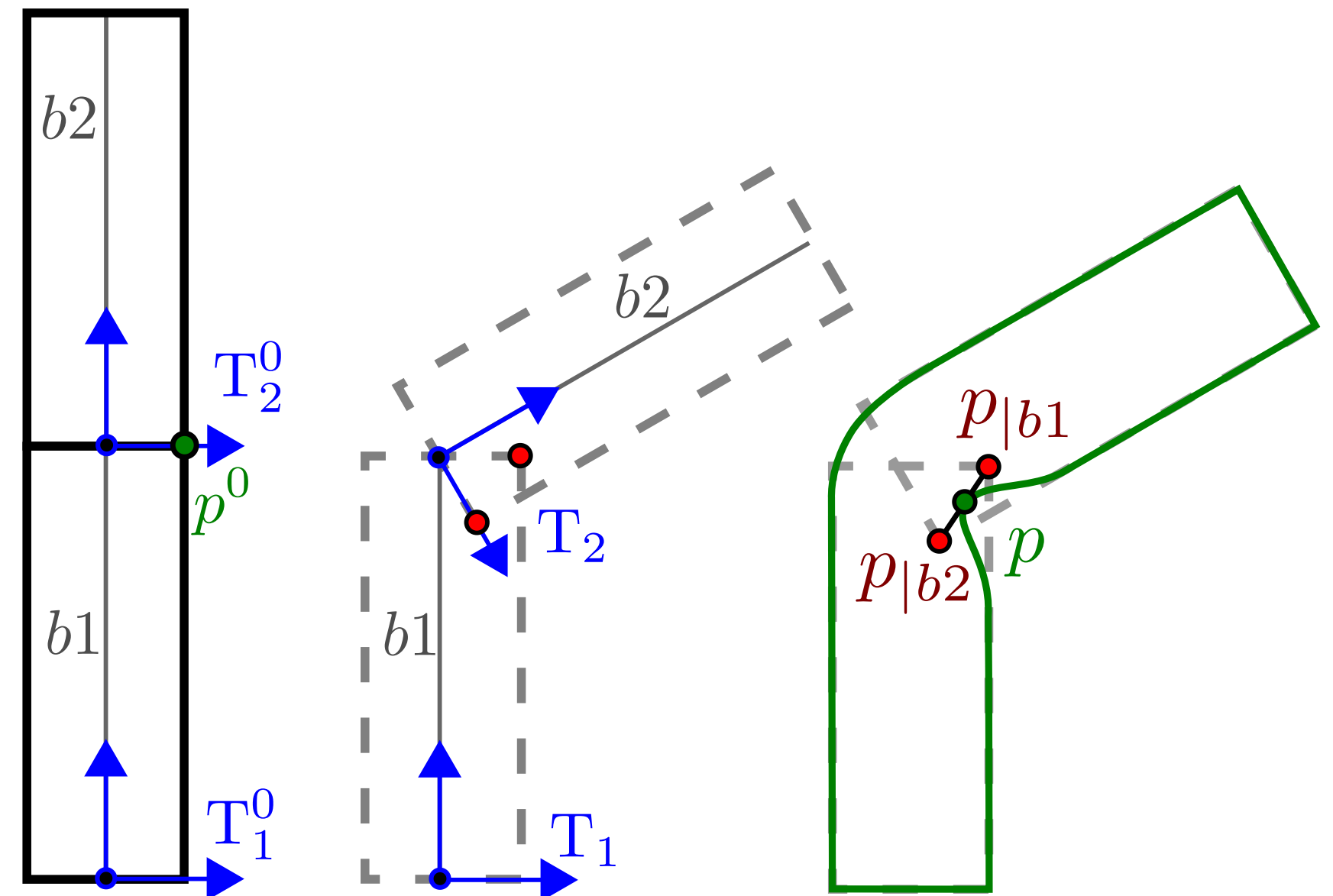
Smooth skinning

46/81

Linear Blend Skinning (LBS): Linear interpolation of positions between associated frames

Example at middle vertex position of a bending cylinder

$$\begin{aligned} p &= 0.5 p_{|b1} + 0.5 p_{|b2} \\ p &= 0.5 \underbrace{T_1 (T_1^0)^{-1}}_{M_1} p^0 + 0.5 \underbrace{T_2 (T_2^0)^{-1}}_{M_2} p^0 \\ p &= (0.5 M_1 + 0.5 M_2) p^0 \end{aligned}$$



Smooth skinning

47/81

Linear Blend Skinning (LBS): Linear interpolation of positions between associated frames

Example at middle vertex position of a bending cylinder

$$\begin{aligned} p &= 0.5 p_{|b1} + 0.5 p_{|b2} \\ p &= 0.5 \underbrace{T_1 (T_1^0)^{-1}}_{M_1} p^0 + 0.5 \underbrace{T_2 (T_2^0)^{-1}}_{M_2} p^0 \\ p &= (0.5 M_1 + 0.5 M_2) p^0 \end{aligned}$$

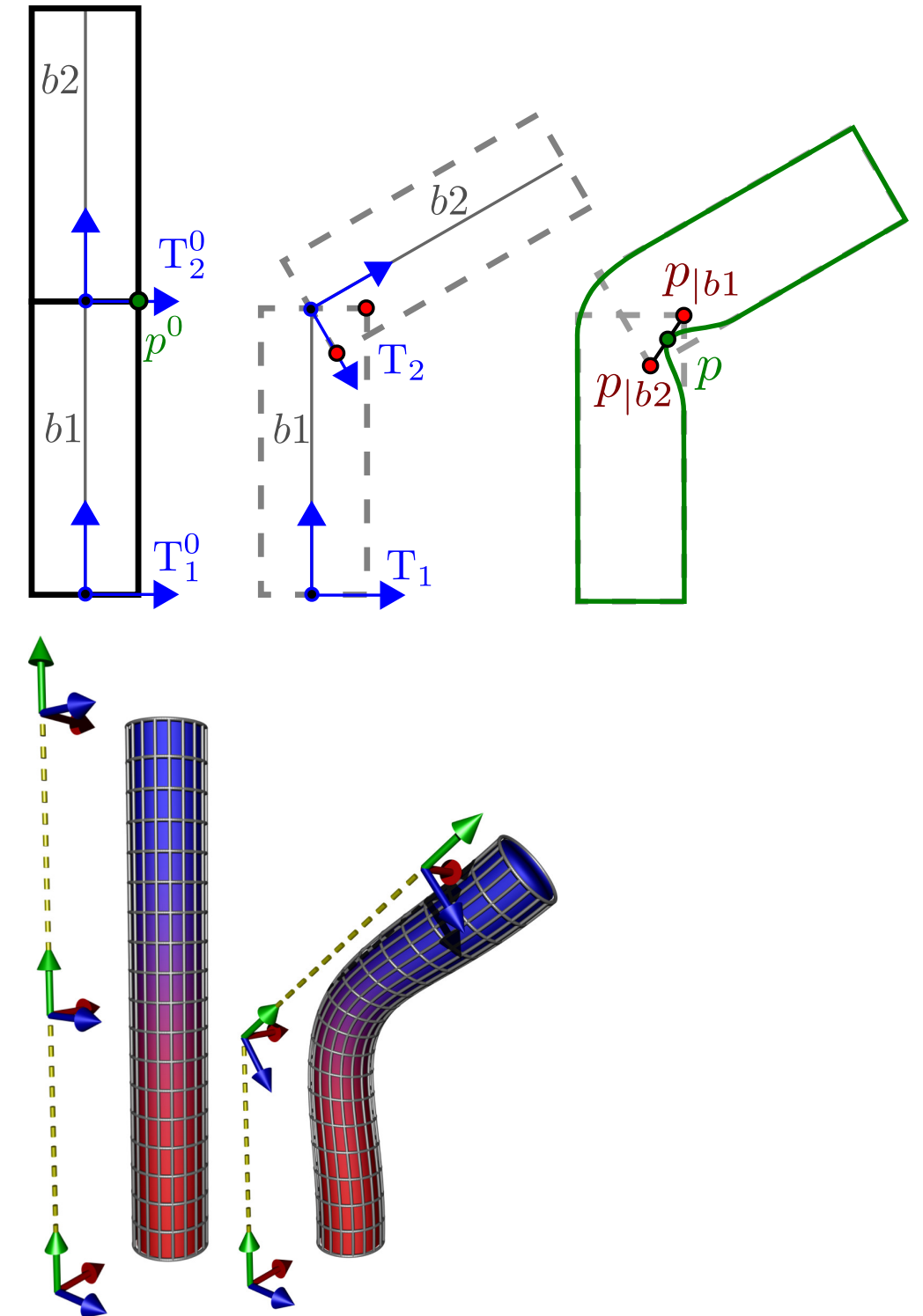
Can be generalized to arbitrary interpolation between two bones

$$p = \alpha p_{|b1} + (1 - \alpha) p_{|b2} = (\alpha M_1 + (1 - \alpha) M_2) p^0$$

α : skinning weights

Can be generalized to any number of bones

$$p = \sum_{k=0}^{N-1} \alpha_k p_{|bk} = \left(\sum_{k=0}^{N-1} \alpha_k M_k \right) p^0, \quad \sum_k \alpha_k = 1$$

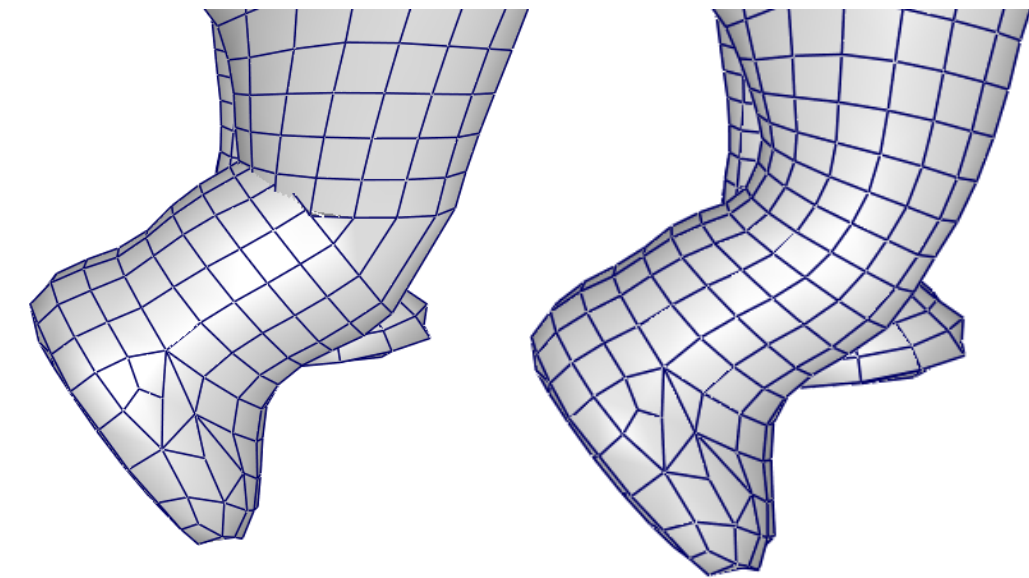
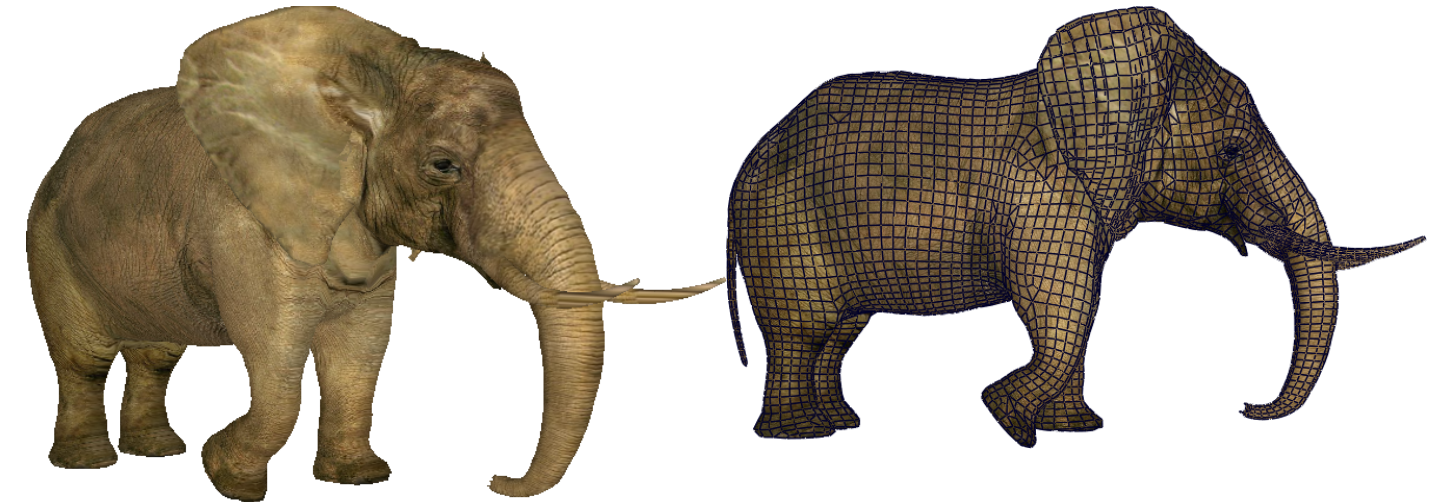


Smooth skinning - Summary

48/81

$$p = \left(\sum_{k=0}^{N-1} \alpha_k M_k \right) p^0 = \left(\sum_{k=0}^{N-1} \alpha_k T_k (T_k^0)^{-1} \right) p^0$$

$\forall k, \alpha_k \in [0, 1], \text{ and } \sum_{k=0}^{N-1} \alpha_k = 1$



The current **standard** for almost all articulated character deformations

- Intuitive deformation
- Controlable shape (*through weights*)
- Fast to compute (GPU compatible)

matrix average, multiplication matrix-vector

⇒ Heavily used in Animation cinema & Video Game

[*Joint-Dependent Local Deformations for Hand Animation and Object Grasping*. Nadia Magnenat-Thalmann, Rochard Laperière, Daniel Thalmann. *Graphics Interface*, 1988]

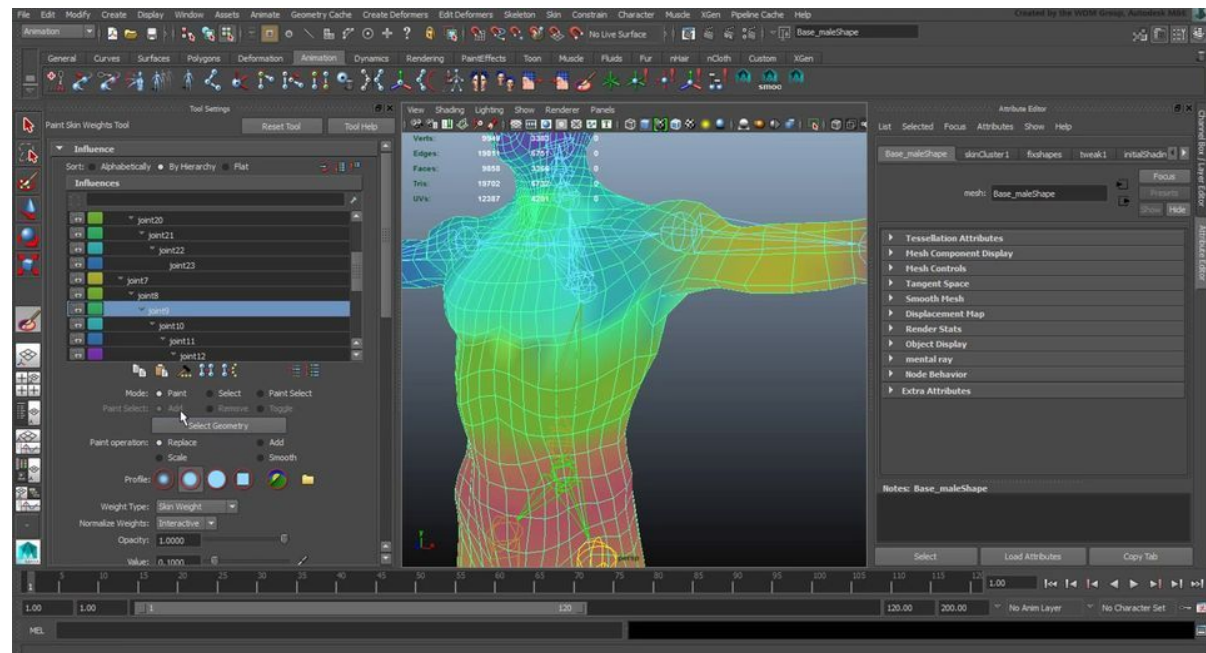
[*Over My Dead, Polygonal Body*. Jeff Lander. *Game Developer Magazine*, 1998]

Skinning weights

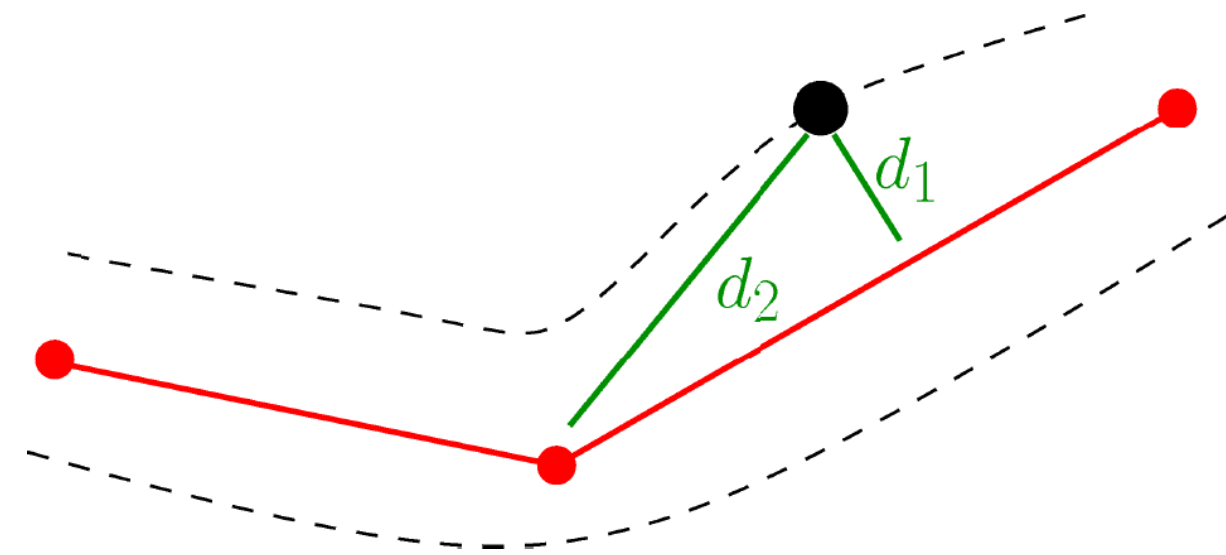
49/81

How to generate skinning weights ?

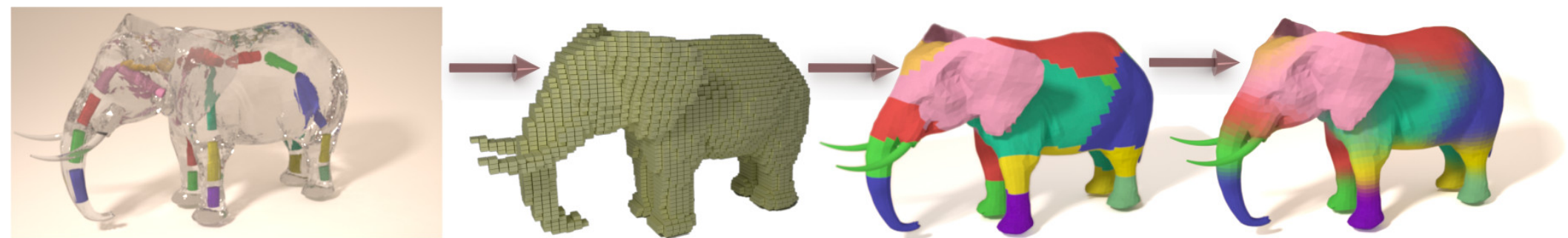
- Paint them manually
- Automatic computation



ex. Using cartesian distances: $\alpha_1 = d_1^{-1} / (d_1^{-1} + d_2^{-1})$



Or using diffusion on the volume/surface



Rigging: Associating bones and skinning weights (or any animation handle) to mesh parts

Skinning: File Format

Unfortunately few standard open format to store skinning animation data

Main open format: Collada (XML), glTF (JSON)

Common software related formats: FBX, Blend, 3DS, ...

```
krissie Maya 7.0 | ColladaMaya v2.04 | FCollada v1.14 Collada Maya Export Options:
bakeTransforms=0;exportPolygonMeshes=1;bakeLighting=0;isSampling=0;
curveConstrainSampling=0;exportCameraAsLookat=0; exportLights=0;exportCameras=1;exportJointsAndSkin=1;
exportAnimations=1;exportTriangles=0;exportInvisibleNodes=0;
exportNormals=1;exportTexCoords=1;exportVertexColors=0;exportTangents=0;
exportTexTangents=0;exportConstraints=0;exportPhysics=0;exportXRefs=0;
dereferenceXRefs=0;cameraXFov=0;cameraYFov=1 file:///E:/maya_projets/girafe_skin/scenes/giranitex_33_bak3.ma 2008-
04-16T21:56:42Z 2008-04-16T21:56:42Z Y_UP 0.040000 0.080000 0.120000 0.160000 0.200000 0.240000 0.280000
0.320000 0.360000 0.400000 0.440000 0.480000 0.520000 0.560000 0.600000 0.640000 0.680000 0.720000 0.760000
0.800000 0.840000 0.880000 0.920000 0.960000 1.000000 1.040000 1.080000 1.120000 1.160000 1.200000 1.240000
1.280000 1.320000 1.360000 1.400000 1.440000 1.480000 1.520000 1.560000 1.600000 1.640000 1.680000 1.720000
1.760000 1.800000 1.840000 1.880000 1.920000 1.960000 2.000000 2.040000 2.080000 2.120000 2.160000 2.200000
2.240000 2.280000 2.320000 2.360000 2.400000 2.440000 2.480000 2.520000 2.560000 2.600000 2.640000 2.680000
2.720000 2.760000 2.800000 2.840000 2.880000 2.920000 2.960000 3.000000 3.040000 3.080000 3.120000 3.160000
3.200000 3.240000 3.280000 3.320000 3.360000 3.400000 3.440000 3.480000 3.520000 3.560000 3.600000 3.640000
3.680000 3.720000 3.760000 3.800000 3.840000 3.880000 3.920000 3.960000 4.000000 4.040000 4.080000 4.120000
4.160000 4.200000 4.240000 4.280000 4.320000 4.360000 4.400000 4.440000 4.480000 4.520000 4.560000 4.600000
4.640000 4.680000 4.720000 4.760000 4.800000 4.840000 4.880000 4.920000 4.960000 5.000000 5.040000 5.080000
5.120000 5.160000 5.200000 5.240000 5.280000 5.320000 5.360000 5.400000 5.440000 5.480000 5.520000 5.560000
5.600000 5.640000 5.680000 5.720000 5.760000 5.800000 5.840000 5.880000 5.920000 5.960000 6.000000 6.040000
6.080000 6.120000 6.160000 6.200000 6.240000 6.280000 6.320000 6.360000 6.400000 6.440000 6.480000 6.520000
6.560000 6.600000 6.640000 6.680000 6.720000 6.760000 6.800000 6.840000 6.880000 6.920000 6.960000 7.000000
7.040000 7.080000 7.120000 7.160000 7.200000 7.240000 7.280000 7.320000 7.360000 7.400000 7.440000 7.480000
7.520000 7.560000 7.600000 7.640000 7.680000 7.720000 7.760000 7.800000 7.840000 7.880000 7.920000 7.960000
8.000000 8.040000 8.080000 8.120000 8.160000 8.200000 8.240000 8.280000 8.320000 8.360000 8.400000 8.440000
8.480000 8.520000 8.560000 8.600000 8.640000 8.680000 8.720000 8.760000 8.800000 8.840000 8.880000 8.920000
8.960000 9.000000 9.040000 9.080000 9.120000 9.160000 9.200000 9.240000 9.280000 9.320000 9.360000 9.400000
9.440000 9.480000 9.520000 9.560000 9.600000 9.640000 9.680000 9.720000 9.760000 9.800000 9.840000 9.880000
9.920000 9.960000 10.000000 10.040000 10.080000 10.120000 10.160000 10.200000 10.240000 10.280000 10.320000
10.360000 10.400000 10.440000 10.480000 10.520000 10.560000 10.600000 10.640000 10.680000 10.720000 10.760000
10.800000 10.840000 10.880000 10.920000 10.960000 11.000000 0.016364 -0.049779 1.881053 0.017034 -0.052046
1.877202 0.018112 -0.055759 1.865879 0.019565 -0.060865 1.847425 0.021359 -0.067310 1.822185 0.023464 -0.075041
1.790502 0.025845 -0.084004 1.752719 0.028471 -0.094146 1.709178 0.031309 -0.105413 1.660224 0.034326 -0.117752
1.606199 0.037199 -0.121199 1.545117 0.040769 -0.125131 1.481219 0.044429 -0.129665 1.415193 0.047599 -0.134756
```

collada.dae

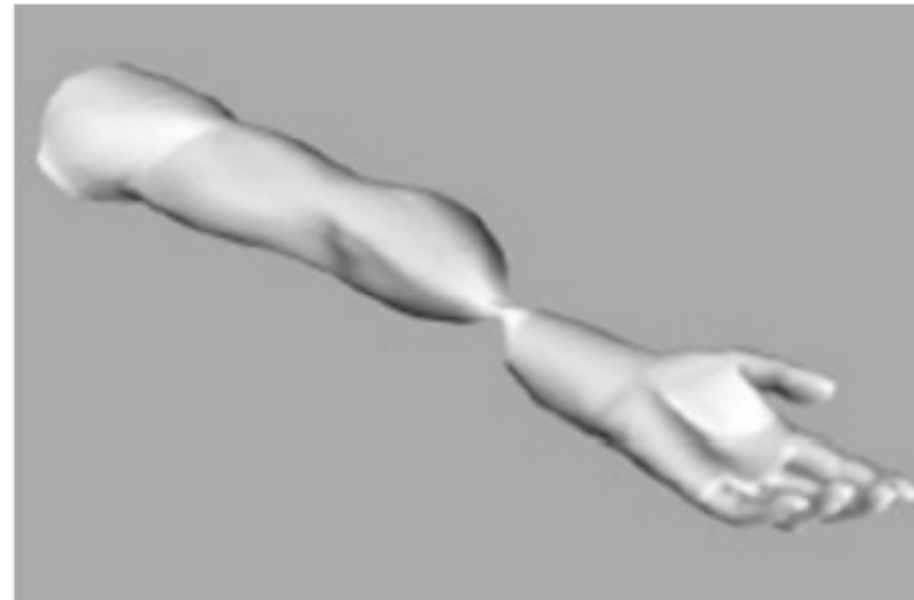
```
{
  "images": [{
    "url": "MarineCv2_color.jpg",
    "uuid": "1F3E9A1C-DDD2-3F4D-80C1-BC5F609DC23B",
    "name": "MarineCv2_color.jpg"
  }],
  "geometries": [{
    "type": "Geometry",
    "uuid": "4181C04A-B75A-3FDC-9A89-89A1D2BCE0AF",
    "data": {
      "uvs":
[[[0.270354,0.313478,0.271947,0.319847,0.275729,0.309655,0.274243,0.3087,0.272832,0.307868,0.269884
,0.310585,0.226132,0.380806,0.222211,0.383513,0.22838,0.387803,0.230469,0.384471,0.220584,0.387519
,0.227778,0.393438,0.236551,0.373639,0.235651,0.37412,0.236463,0.379508,0.237612,0.379488,0.239952
,0.371796,0.240752,0.379751,0.245983,0.382396,0.244072,0.372847,0.029897,0.380277,0.026492,0.37821
7,0.024403,0.381866,0.027378,0.383736,0.23588,0.388524,0.23554,0.386494,0.234758,0.383916,0.050509
,0.560407,0.047071,0.557888,0.045167,0.558824,0.048244,0.563377,0.064365,0.566106,0.055853,0.56226
3,0.05439,0.567096,0.064598,0.572284,0.04127,0.560623,0.045026,0.567068,0.051922,0.572939,0.058767
,0.576525,0.037322,0.561994,0.040429,0.570751,0.047131,0.578106,0.034887,0.563887,0.036649,0.57368
8,0.041729,0.581221,0.051832,0.587997,0.054417,0.583009,0.033166,0.564334,0.033893,0.574381,0.0382
76,0.583055,0.046687,0.590068,0.029228,0.564585,0.030582,0.575915,0.035745,0.587145,0.043417,0.594
797,0.012878,0.575167,0.025786,0.576822,0.024107,0.563407,0.011428,0.55775,0.019835,0.589523,0.031
426,0.590171,0.056273,0.542792,0.059073,0.545878,0.065176,0.5447,0.063035,0.542068,0.048447,0.5468
91,0.053936,0.549727,0.04472,0.552324,0.048873,0.553322,0.055773,0.539823,0.044725,0.545943,0.0644
16,0.538785,0.040454,0.553447,0.054384,0.536842,0.041786,0.545212,0.065807,0.534961,0.037489,0.554
075,0.049928,0.535922,0.039805,0.54519,0.035552,0.554559,0.046737,0.533753,0.037925,0.544124,0.033
877,0.554064,0.043337,0.529892,0.034539,0.543152,0.030422,0.55413,0.04709,0.510318,0.042159,0.5067
9,0.026639,0.514508,0.031141,0.519839,0.038545,0.525958,0.02404,0.532227,0.030178,0.540512,0.01661
9,0.542977,0.02613,0.552544,0.019243,0.52681,0.072806,0.54418,0.068036,0.541536,0.069568,0.544295,
0.072537,0.546332,0.074457,0.542001,0.070694,0.540527,0.075113,0.538587,0.072083,0.53632,0.070949,
0.565762,0.071836,0.562397,0.067553,0.560436,0.064941,0.562816,0.080185,0.557515,0.078931,0.555483
,0.076466,0.562489,0.076226,0.565704,0.058861,0.594811,0.064195,0.589419,0.069791,0.550726,0.06658
1,0.555646,0.071219,0.555536,0.07257,0.547845,0.061829,0.558444,0.058373,0.559956,0.058247,0.55486
4,0.053888,0.556612,0.529304,0.679074,0.528585,0.691852,0.525233,0.680917,0.521958,0.67057,0.00535
1,0.580703,0.020611,0.59504,0.002847,0.554083,0.040637,0.597389,0.01118,0.540222,0.054475,0.599187
,0.046209,0.601951,0.073623,0.5477,0.073842,0.556482,0.06595,0.597735,0.05175,0.602881,0.062732,0.
603138,0.513246,0.652152,0.507764,0.643571,0.505233,0.633772,0.522766,0.650073,0.06824,0.506816,0.
075348,0.508814,0.073568,0.50543,0.068519,0.503123,0.062065,0.55231,0.067781,0.548426,0.055314,0.5
17723,0.050992,0.513294,0.062031,0.506743,0.064039,0.51368,0.068682,0.511233,0.061823,0.58071,0.08
```

.json

Linear Blend Skinning - Limitations

51/81

- Non-trivial rigging settings
- Artifacts for large rotations: Candy wrapper, Collapsing elbow

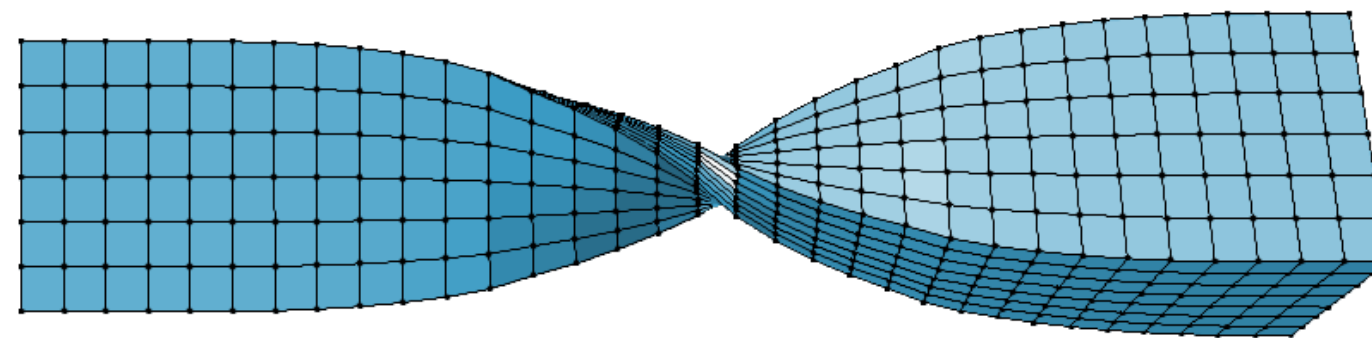


Candy wrapper



Collapsing elbow

Linear blending between rigid transformation matrices



Character Animation

Skeletal Animation

Skeleton structure

53/81

Characteristics

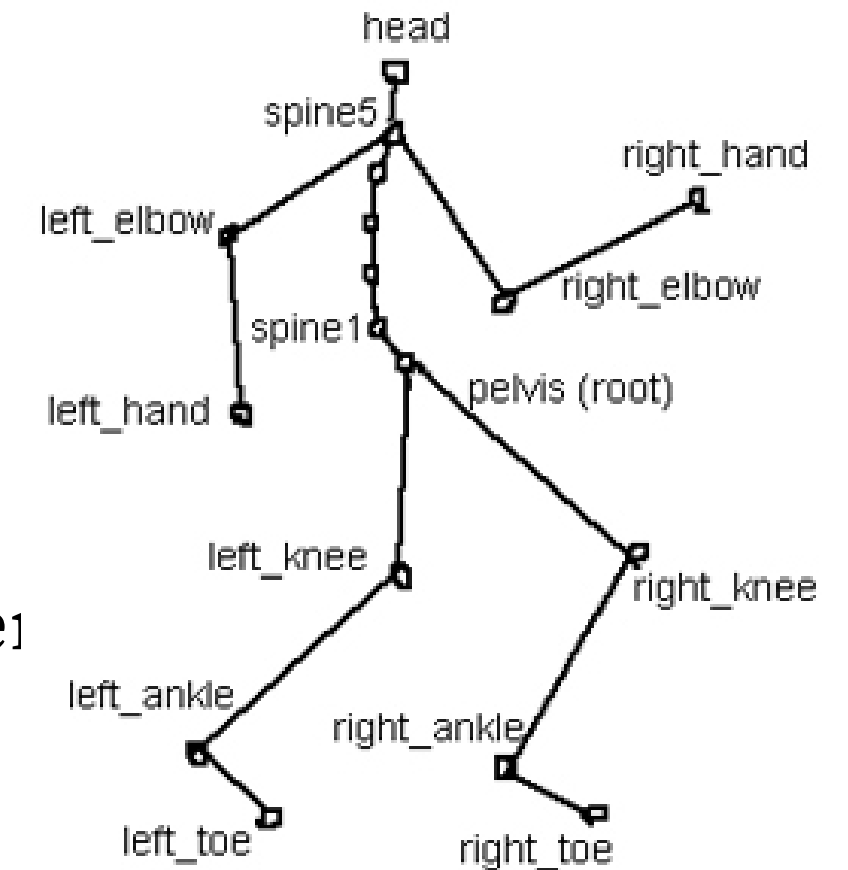
- Hierarchical representation

All children follow the transformation of the parent

Need to define a root: usually at the hips/pelvis

- Convenient to express relative transformation with respect to the parent

ex. Ankle is at 20° w/r knee



Converting local to global frames/joint coordinates

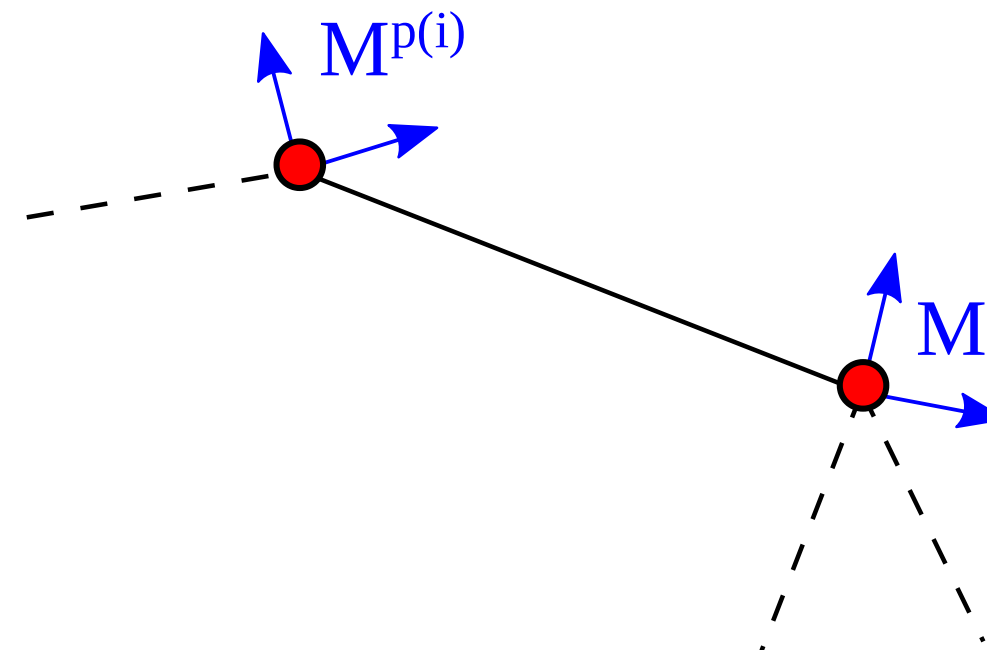
- With 4x4 matrices M

$$M_{global}^i = M_{global}^{p(i)} M_{local}^i$$

- With translation t , rotation R

$$R_{global}^i = R_{global}^{p(i)} R_{local}^i$$

$$t_{global}^i = t_{global}^{p(i)} + R_{global}^{p(i)} t_{local}^i$$



Encoding hierarchical skeleton

54/81

- Simplest encoding based on index within vector

Geometry=[M0, M1, M2, M3, M4, M5]

Parent = [-1,0,1,1,0,4]

- Convert local coordinates to global coordinates

local (Geometry) <- std::vector of rotation (r), translation (p)

global (Geometry) <- std::vector of rotation (r), translation (p)

```
global[0] = local[0];
```

```
for(size_t k=1; k<N; ++k)
```

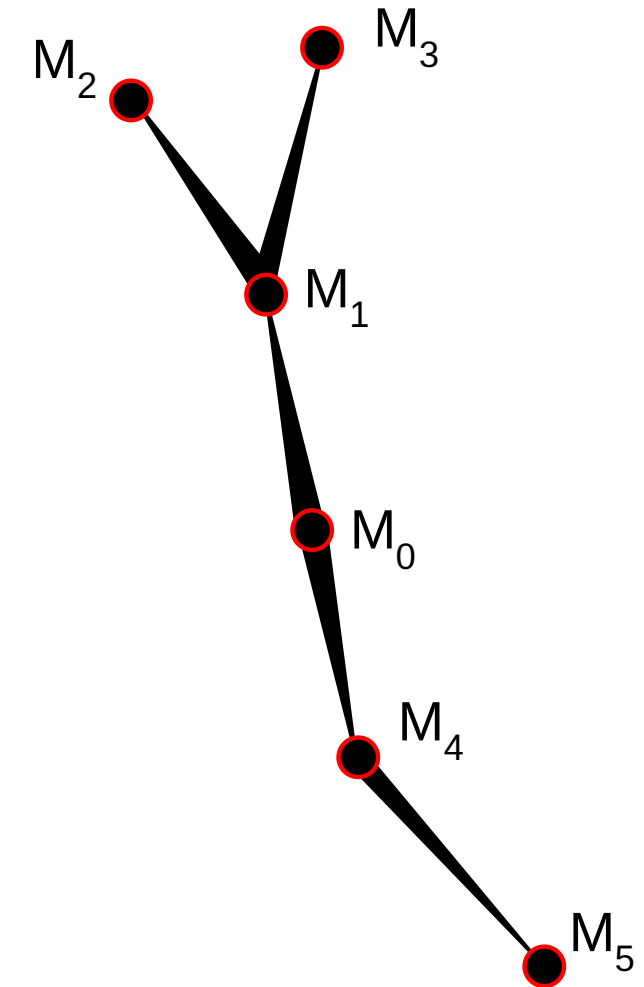
```
{
```

```
    int parent = Parent[k];
```

```
    global[k].r = global[parent].r * local[k].r;
```

```
    global[k].p = global[parent].r * local[k].p + global[parent].p;
```

```
}
```



Synthesizing and controlling skeleton animation

Blending skeleton animation

56/81

Pre-store several looping animation

Blend between animation for transition

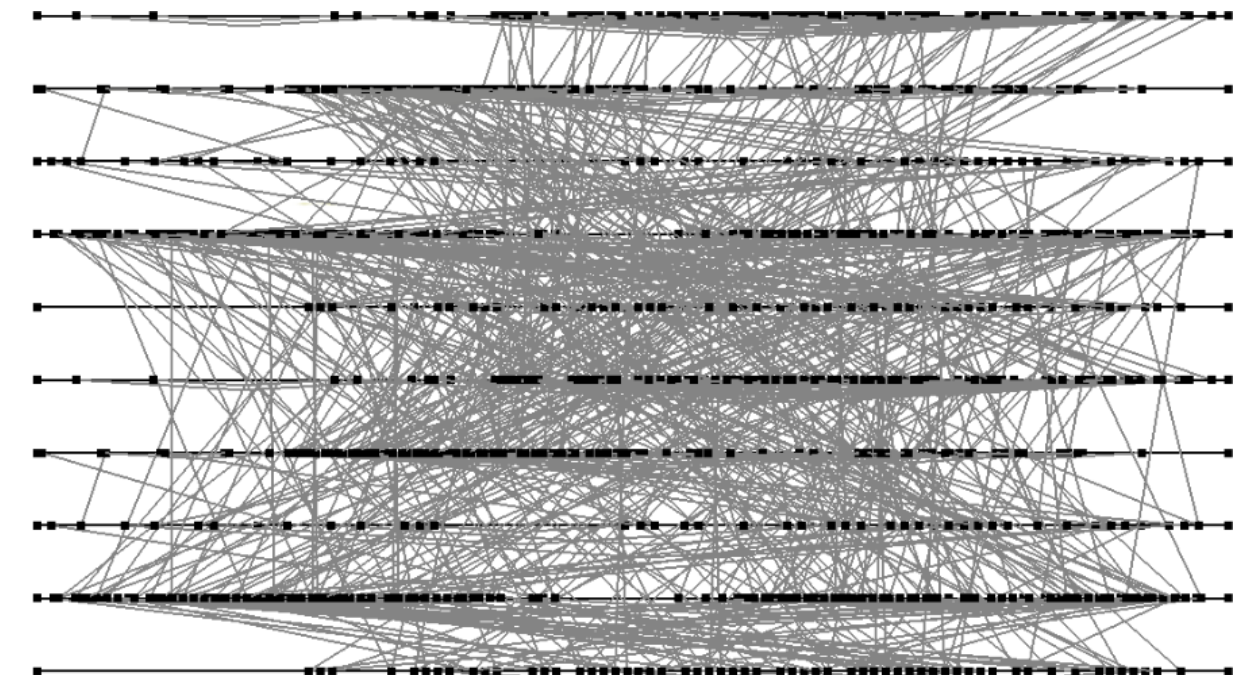
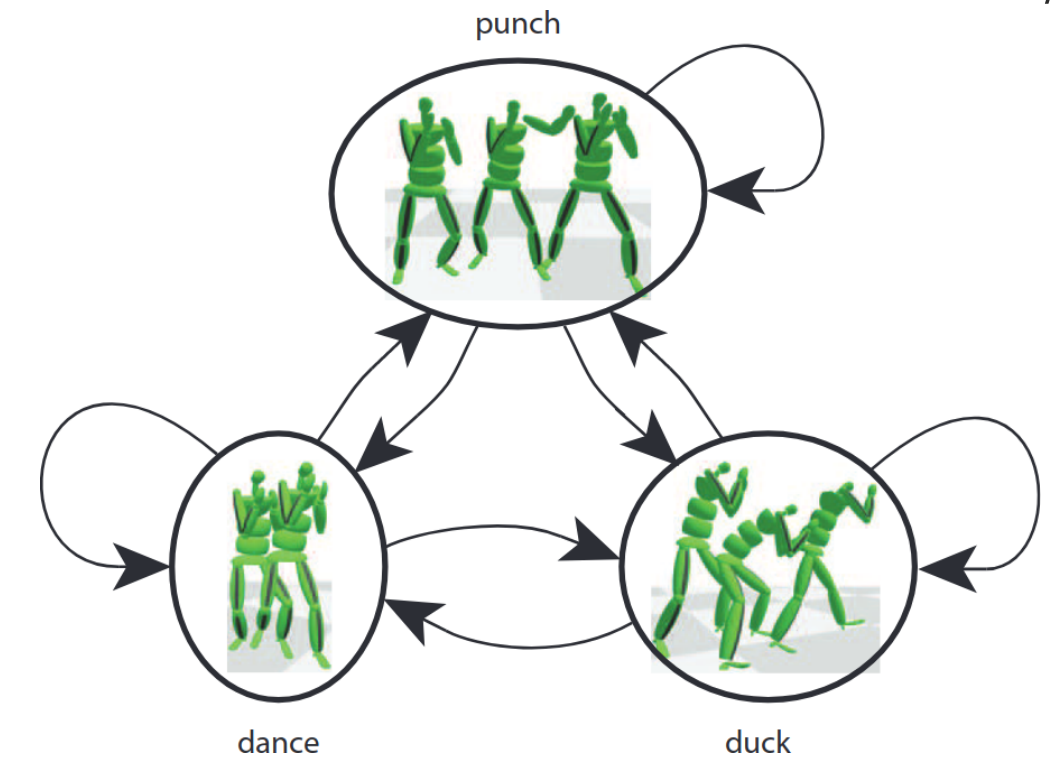
```
three.js - Skeletal Animation Blending (model from realitymelt.com)  
camera orbit/zoom/pan with left/middle/right mouse button  
Note: crossfades are possible with blend weights being set to (1,0,0), (0,1,0) or (0,0,1)
```

Motion graphs

57/81

Also called Move Trees (highly used in video games)

- Stores multiple precomputed animation
Manually design, motion capture, etc
- Find optimal transitions between different motions
- [Mizuguchi et al., Data driven motion transitions for interactive games, EG short paper, 2001]
- [Kovar et al., Motion Graphs, ACM SIGGRAPH 2002]
- [Heck and Gleicher, Parametric Motion Graphs, ACM SIGGRAPH 2007]

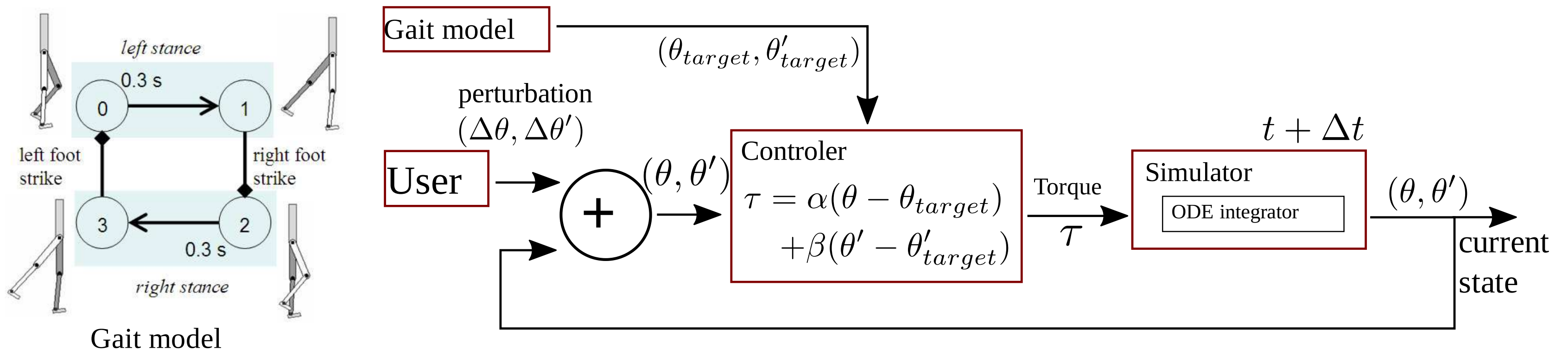
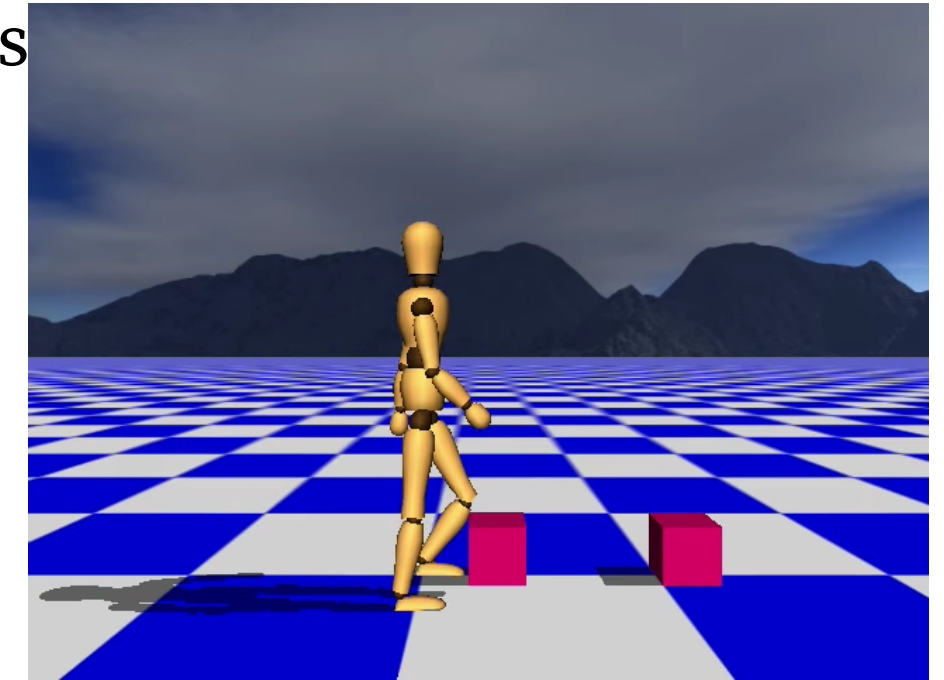


Controllers

58/81

Mix between predefined motions and physics → allow user perturbations

- 1 - Define target $(\theta_{target}(t), \theta'_{target}(t))$
pre-defined finite state machine (Gait model)
- 2 - Add user perturbation to the current state
- 3 - Use proportional derivative controllers to compute joint torque τ
- 4 - Integrate torque using rigid body simulator
- 5 - Iterate



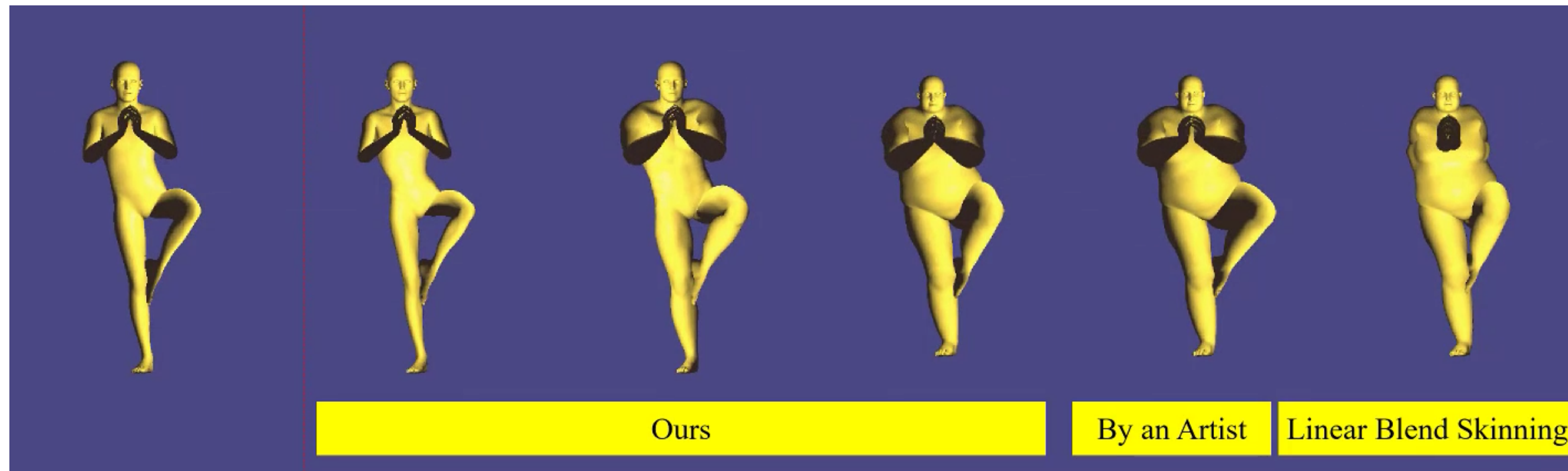
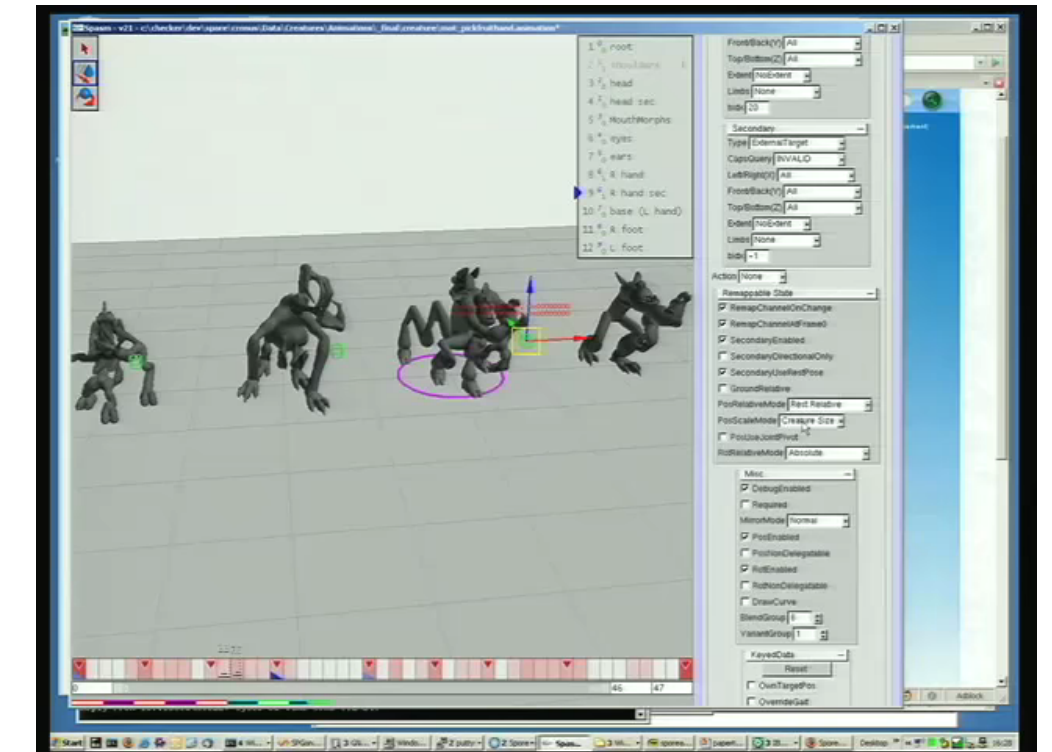
[M. Raibert and J. Hodgins. *Animation of Dynamic Legged Locomotion*, ACM SIGGRAPH 2001]

[K. Yin et al., *SIMBICON: Simple Biped Locomotion Control*, ACM SIGGRAPH 2007]

Motion transfert

59/81

- Local coordinates mapping from skeletal motions
 - [C. Hecker et al., Real-time Motion Retargeting to Highly Varied User-Created Morphologies, SIGGRAPH 2008] (Spore)
- Including shape morphology
 - [Z. Liu et al., Surface based Motion Retargeting by Preserving Spatial Relationship, MIG 2018]

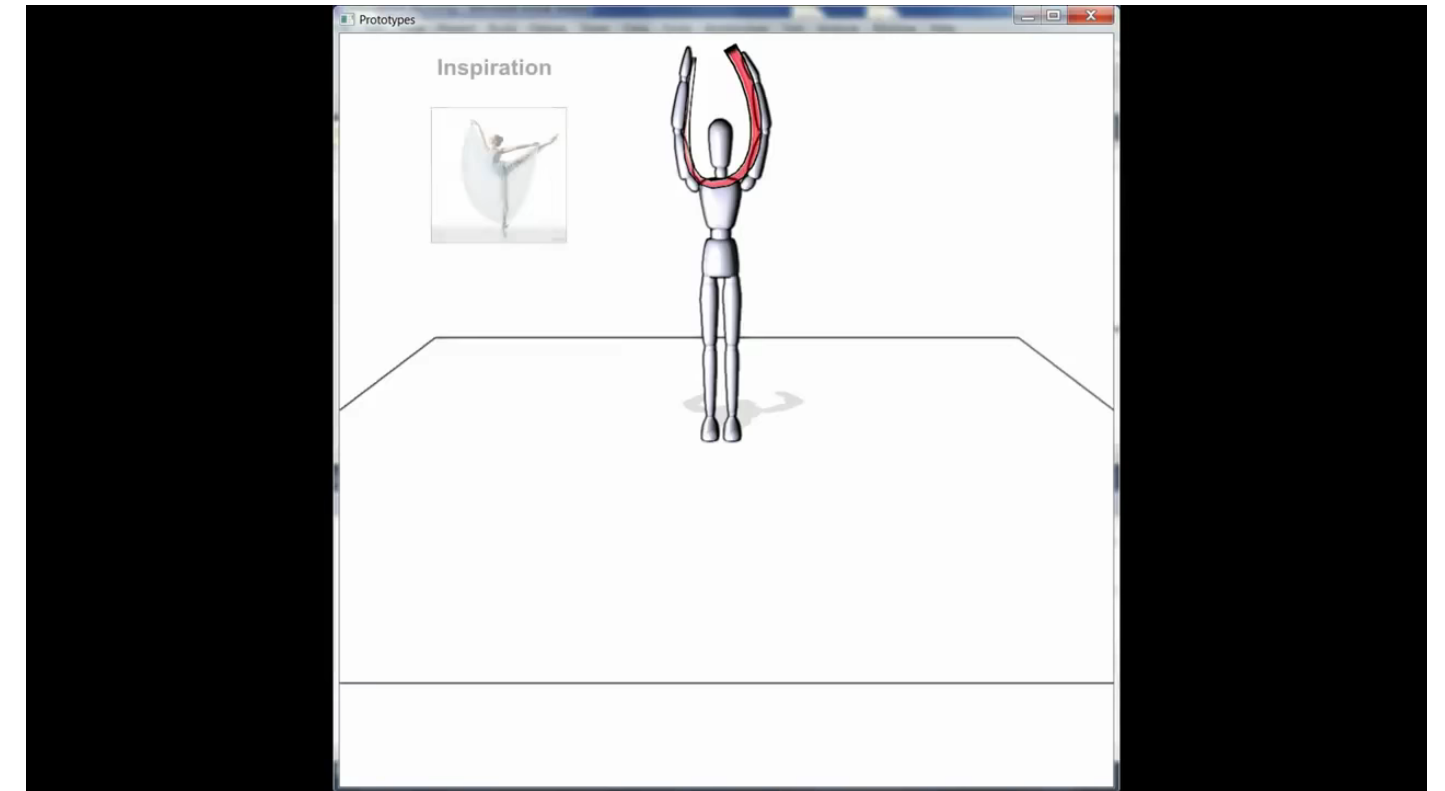


Animation design

60/81

Based on the *Line of Action*

- [Guay et al., The Line of Action: an Intuitive Interface for Expressive Character Posing, ACM SIGGRAPH Asia 2013]
- [Guay et al., Space-time sketching of character animation, ACM SIGGRAPH 2015]
- [Choi et al., SketchiMo: Sketch-Based Motion Editing for Articulated Characters, ACM SIGGRAPH 2016]



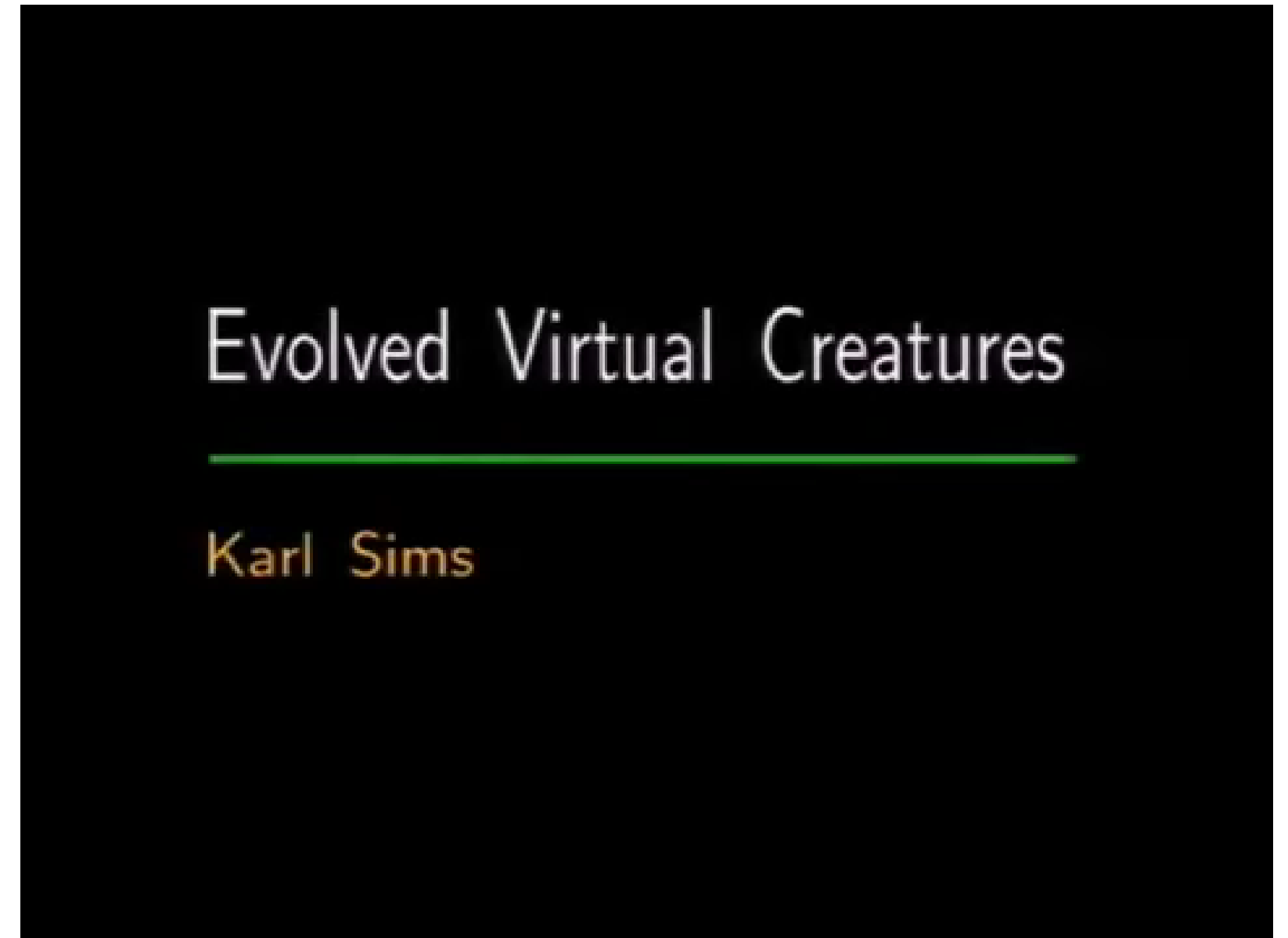
0:00 / 0:35

Automatic synthesis of skeletal animation

61/81

Seminal works

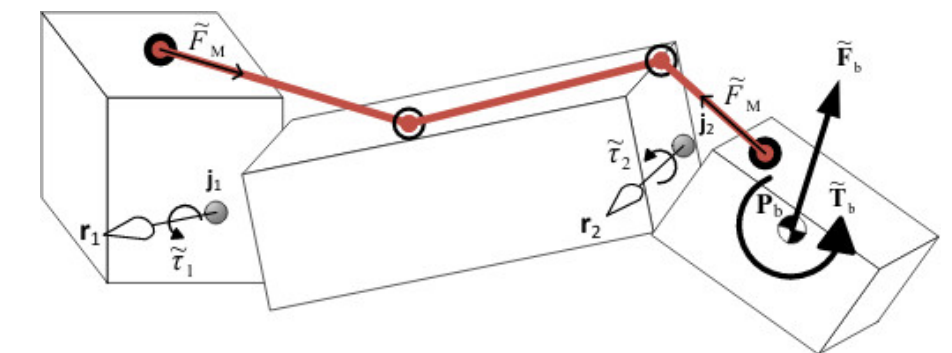
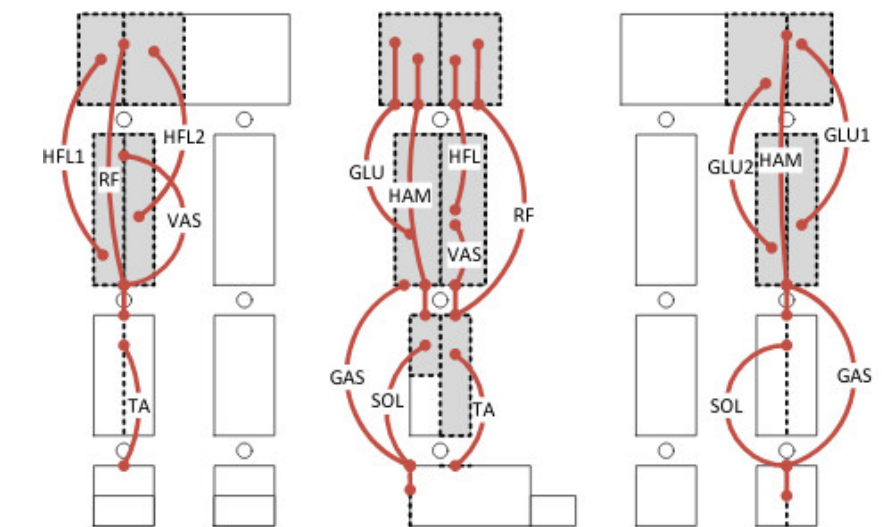
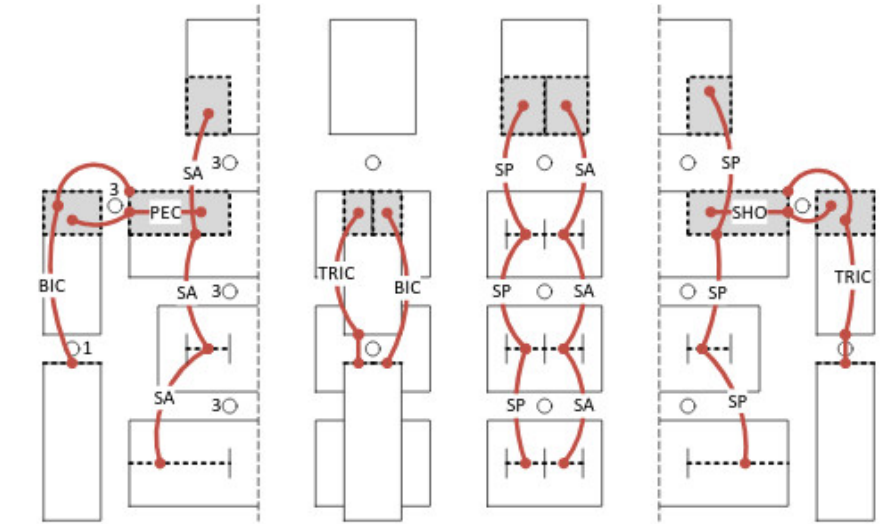
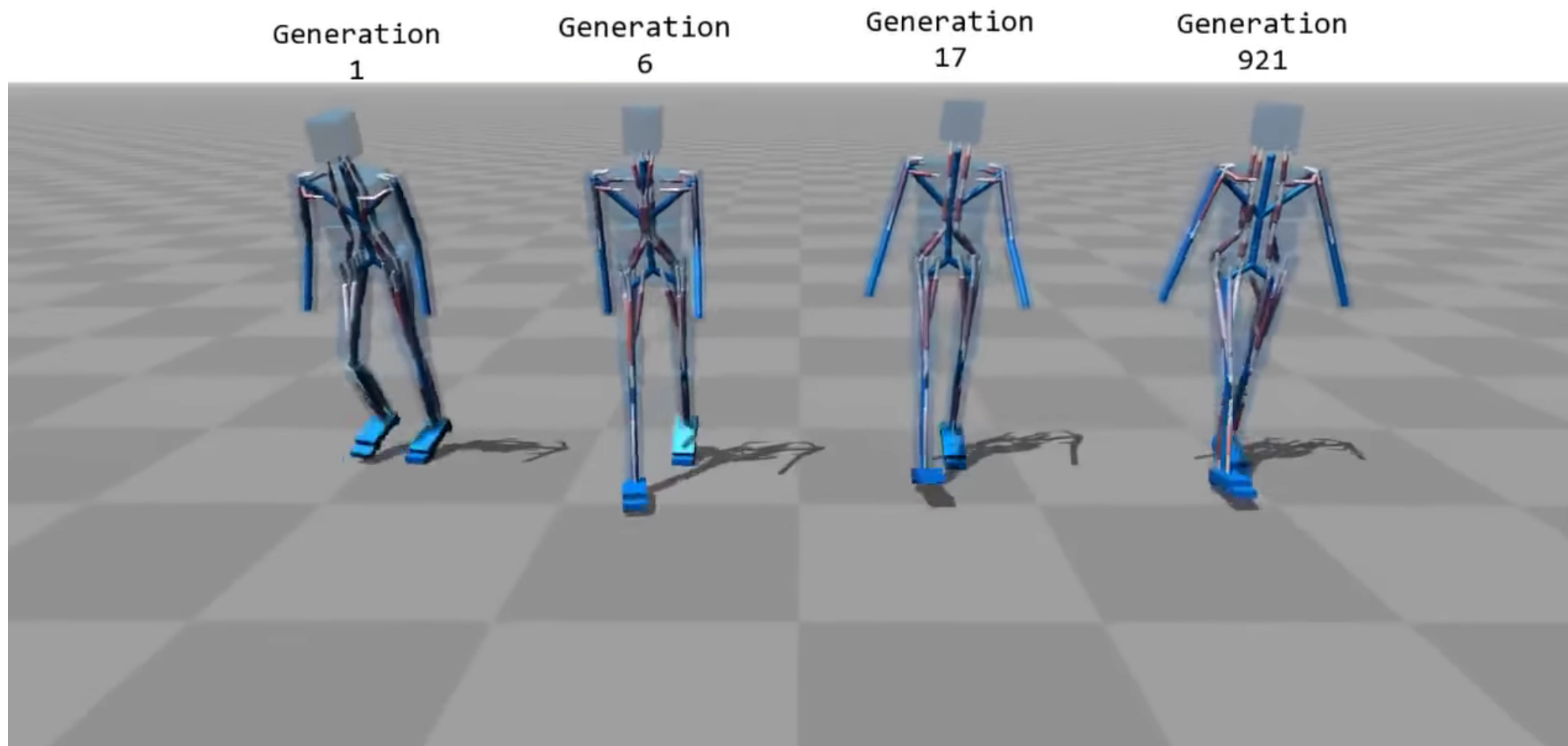
- [Evolving Virtual Creatures. Karl Sims. SIGGRAPH 1994]
- [Automated Learning of Muscle-Actuated Locomotion Through Control Abstraction. Radek Grzeszczuk and Demetri Terzopoulos. SIGGRAPH 1995]
- *Optimization toward objective function coupled with rigid bodies simulations*
- *Morphological variation from genetic algorithm*



Optimizing muscle activation

- Take into account simple biomechanical model
- Optimize sequence of activation via reinforcement learning

[*Flexible Muscle-Based Locomotion for Bipedal Creatures*, T. Geijtenbeek et al. SIGGRAPH Asia 2013.]



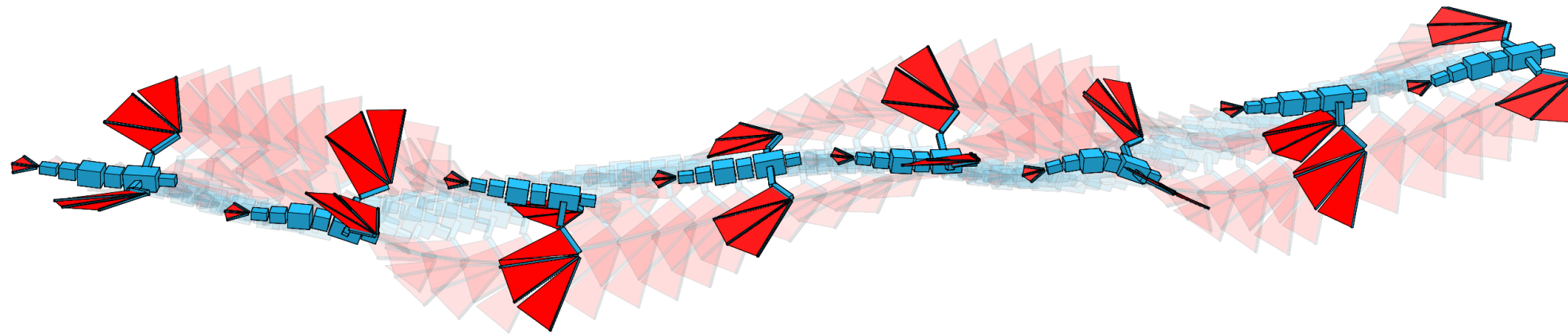
Use of deep learning

63/81

Deep reinforcement learning for complex optimization

Learn muscle activation

[Won et al. *How to Train Your Dragon: Example-Guided Control of Flapping Flight*, ACM SIGGRAPH Asia 2017]



Deep learning for real-time motion control

Learn phase of the motion cycle.

Use large data base of motion capture data

[Holden et al., *Phase-Functioned Neural Networks for Character Control*, ACM TOG 2017]

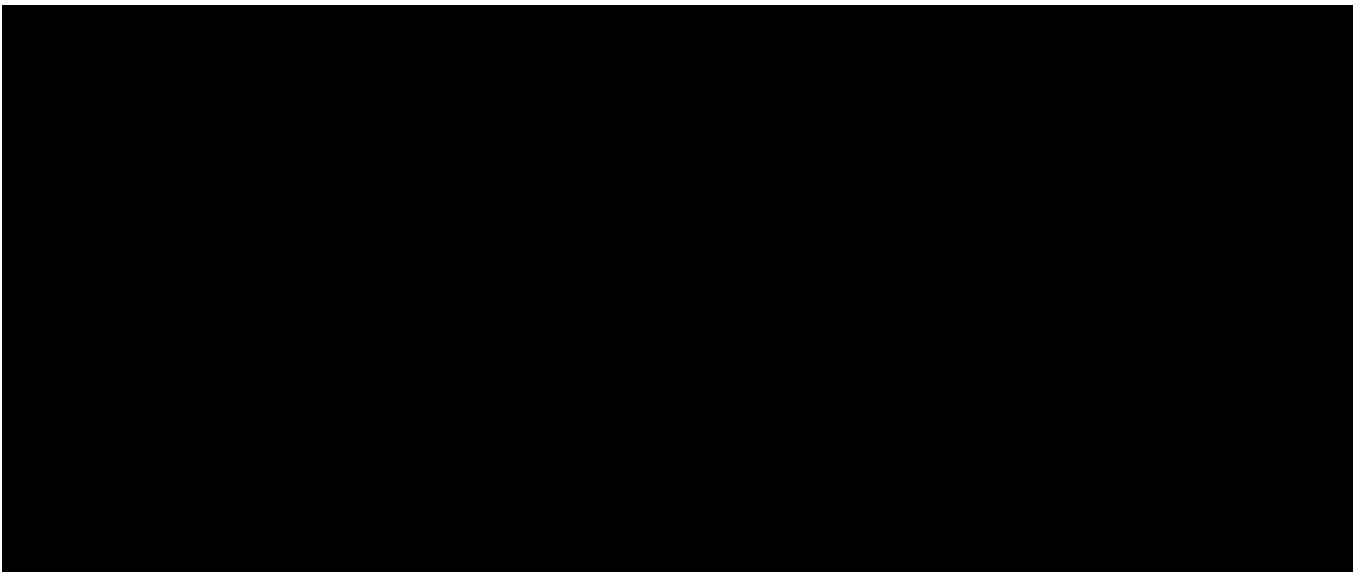
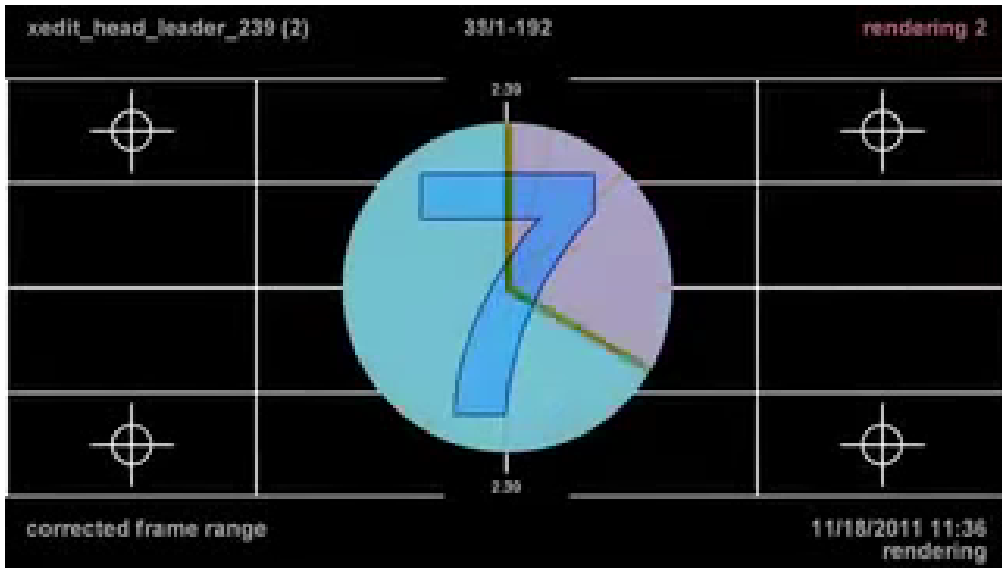
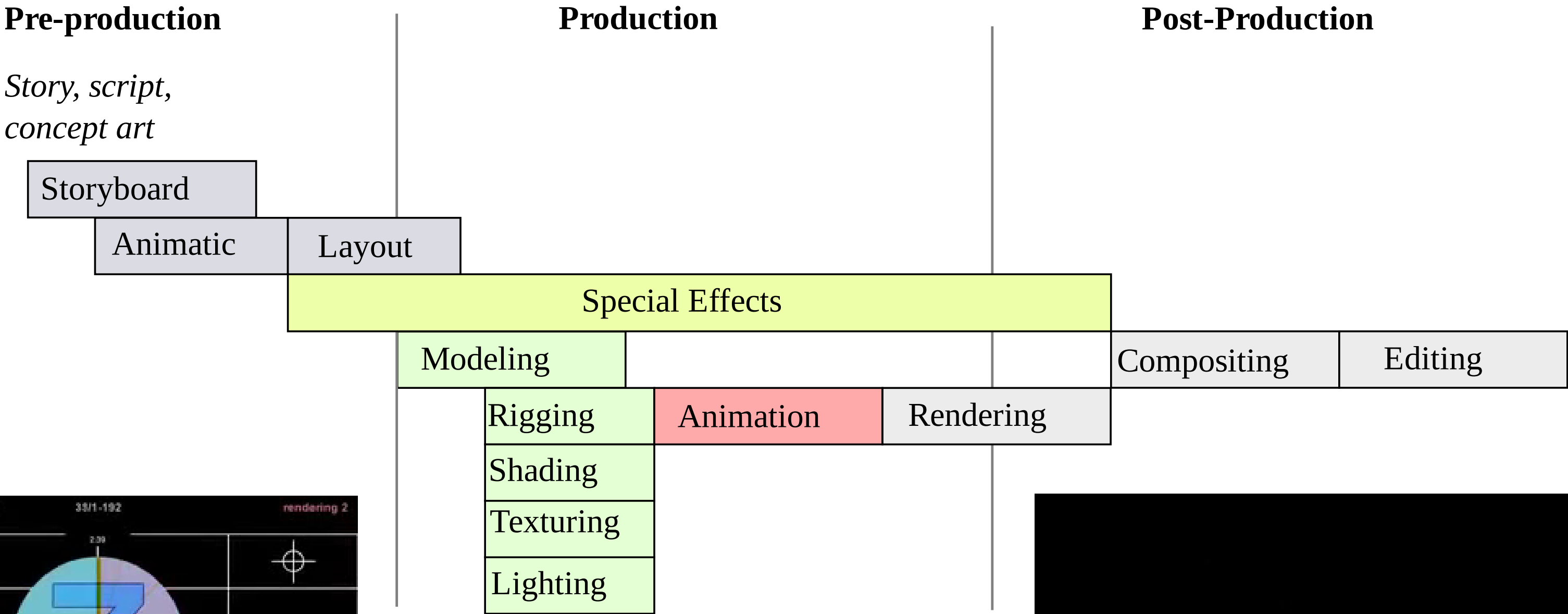


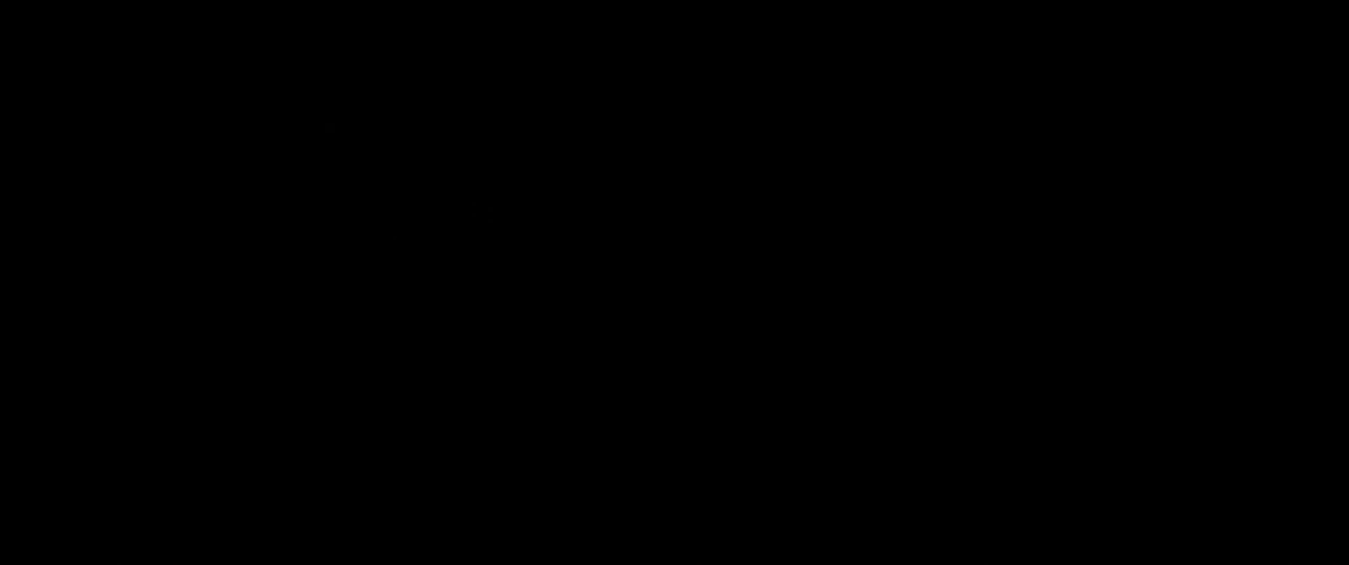
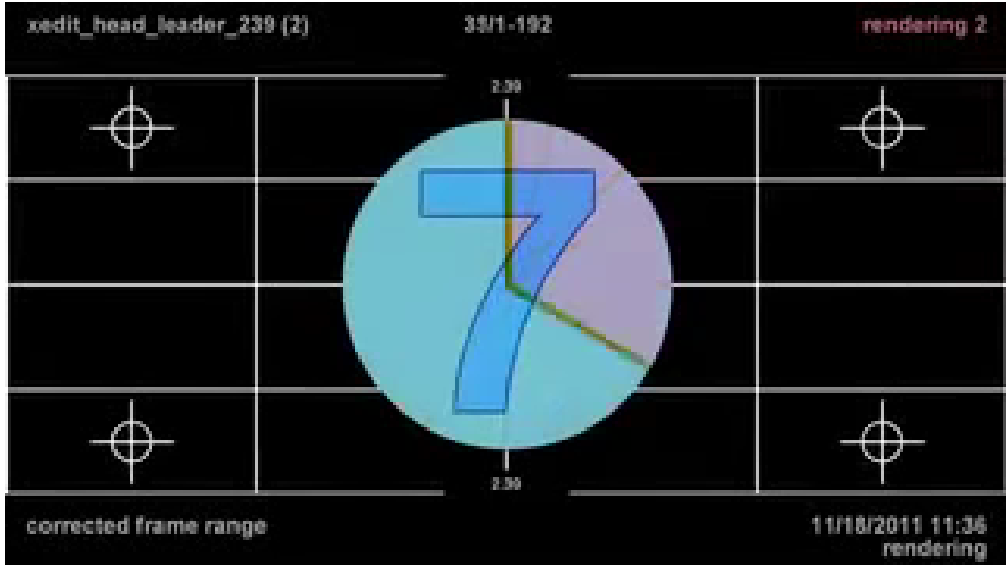
Production Pipeline

Case of Animation Cinema production

Production Pipeline

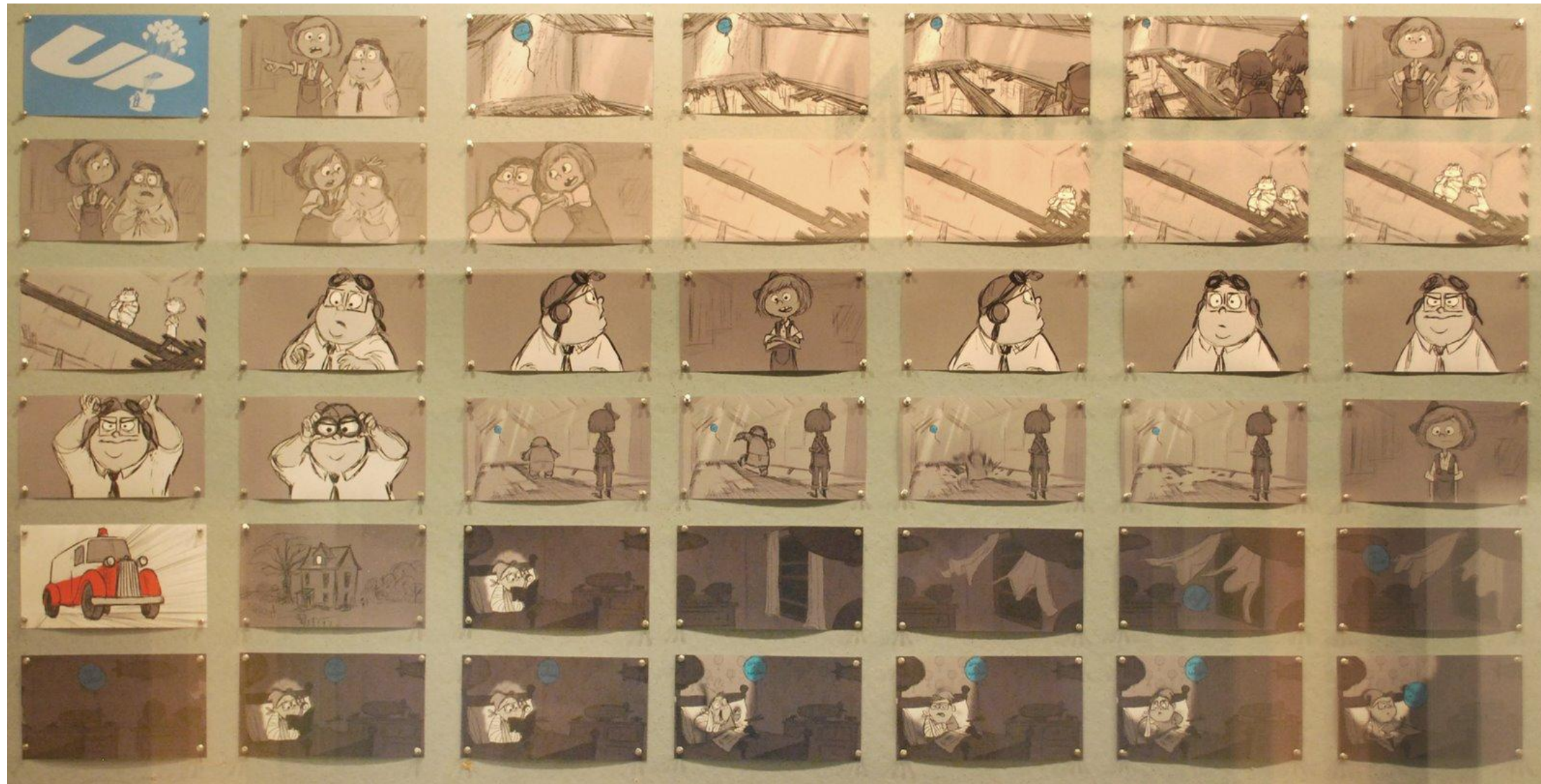
Fundamental structure for VFX/animation movie production.





Story-board

66/81



Few 2D drawings: express the story

Before mostly non-technical - artistic/creative based

Curently Strong increase of Storytelling-related researchs in CG

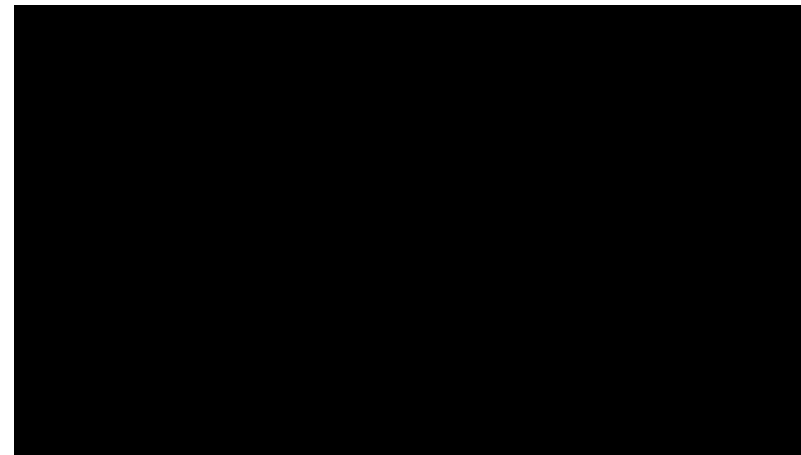
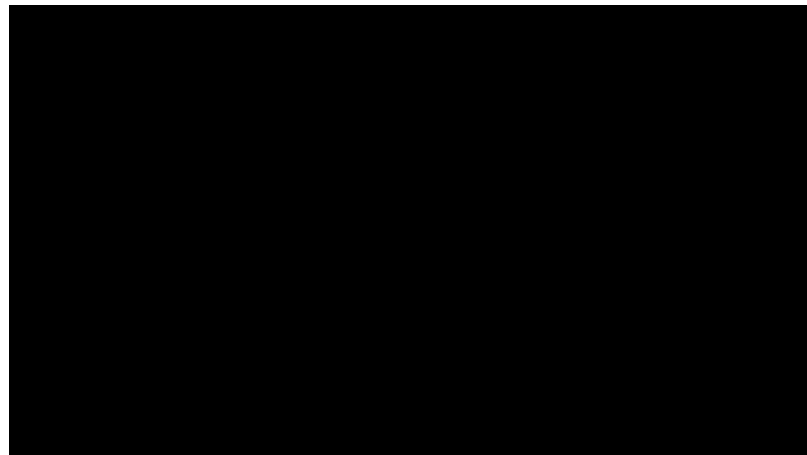
ex. *Disney Research Studios, Anima team at Inria*

Animatic

67/81

≈ Animated story-board, Various format

Rough sense of timing, visual, action: details not necessarily followed precisely in the final version.



Layout: Moving into 3D

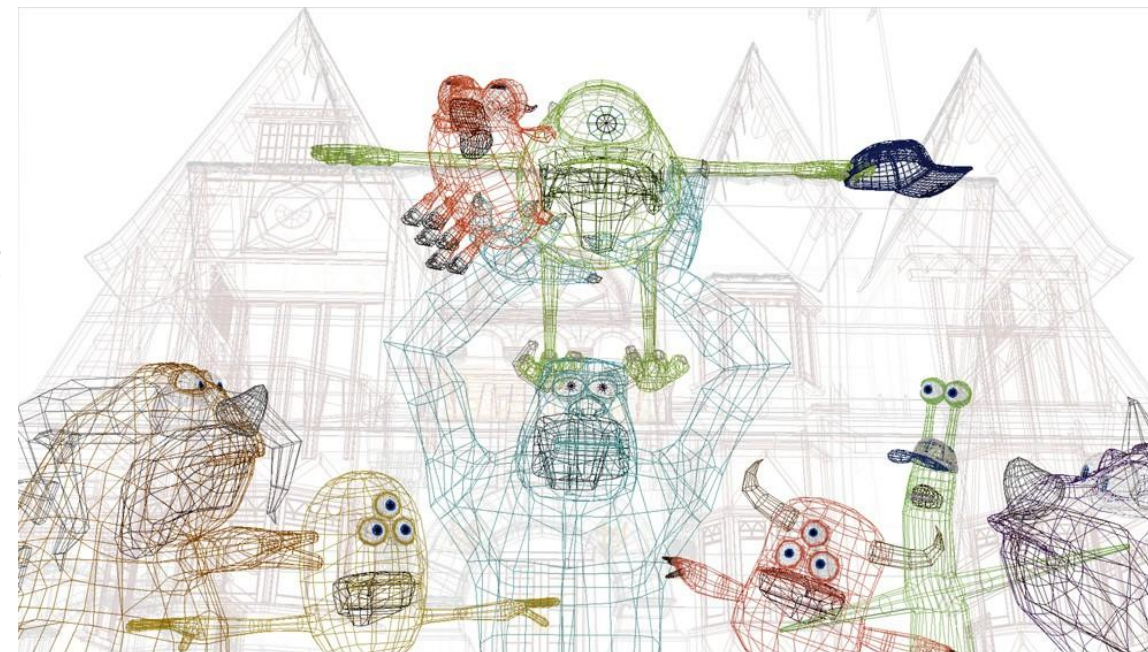
68/81

1st step: Rough 3D modeling and placement of

- Camera: visual field, perspective
3D much more constrained than 2D drawings
- Shape volumes, continuity
No details: face, etc.
Choice between 3D/2D elements



"MONSTERS UNIVERSITY" Progression Image 1 of 6: STORY
©2013 Disney•Pixar. All Rights Reserved.



"MONSTERS UNIVERSITY" Progression Image 3 of 6: MODELING
©2013 Disney•Pixar. All Rights Reserved.



"MONSTERS UNIVERSITY" Progression Image 4 of 6: LAYOUT
©2013 Disney•Pixar. All Rights Reserved.

3D Modeling

69/81

In production

- Polygonal mesh modeling

Coarse to fine approach

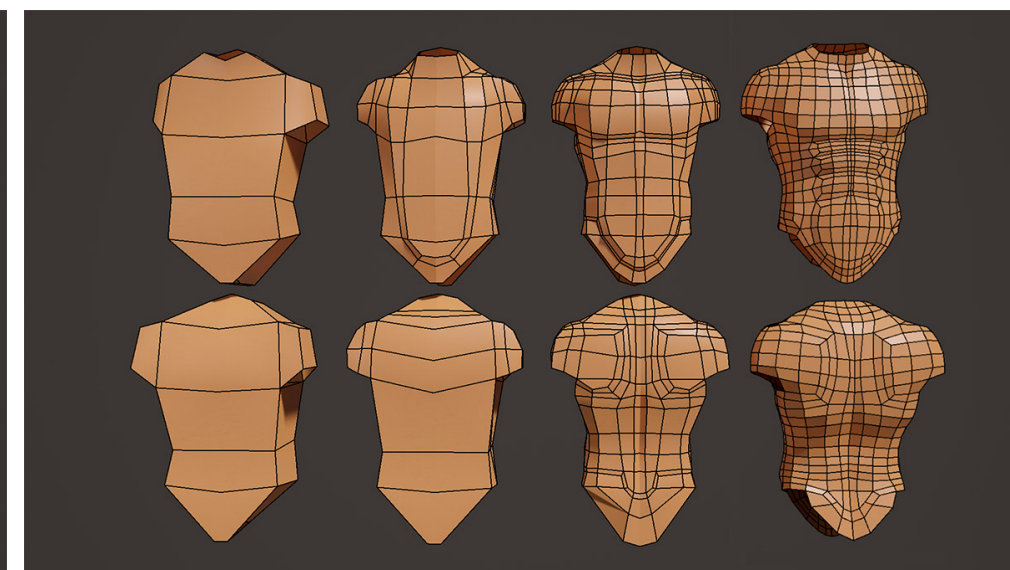
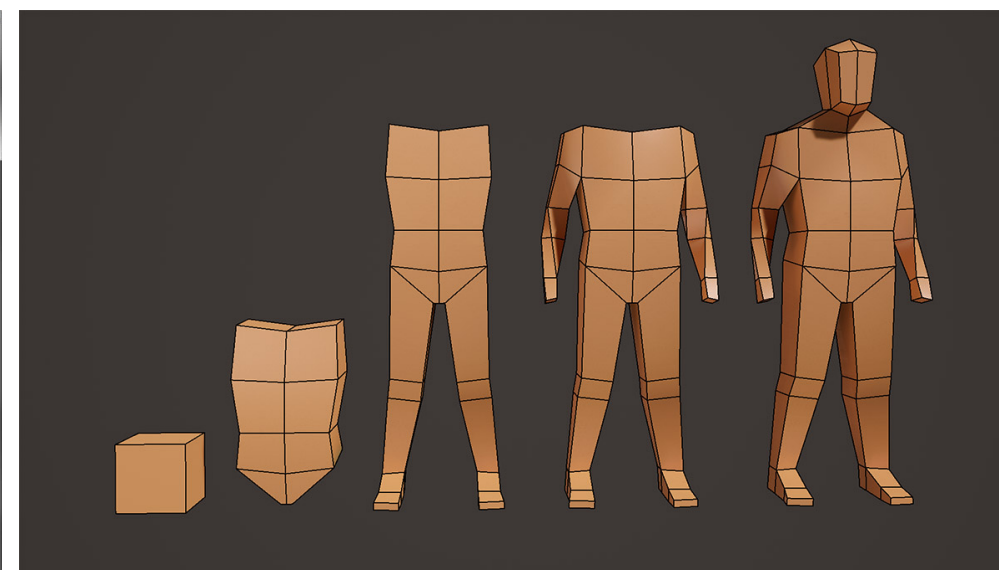
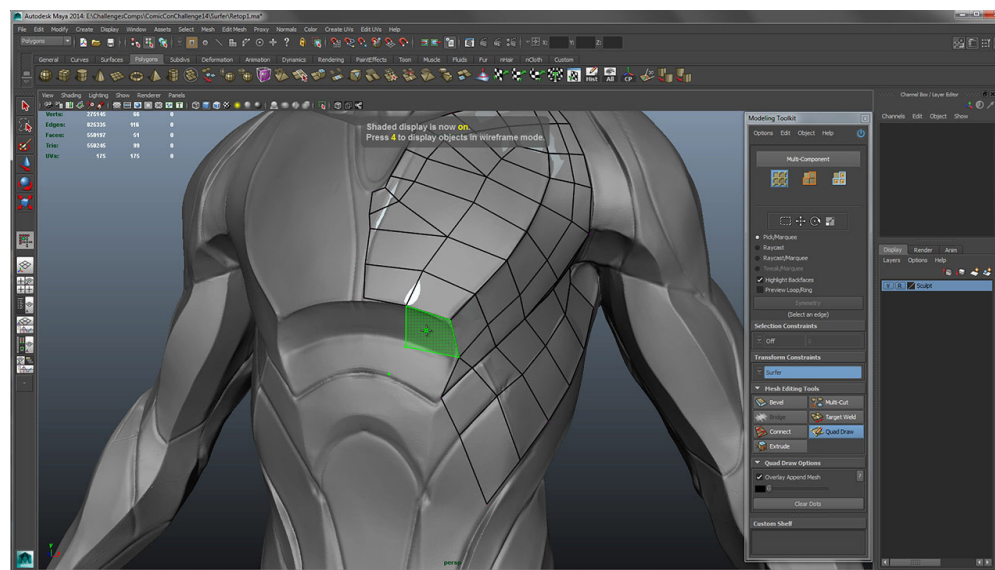
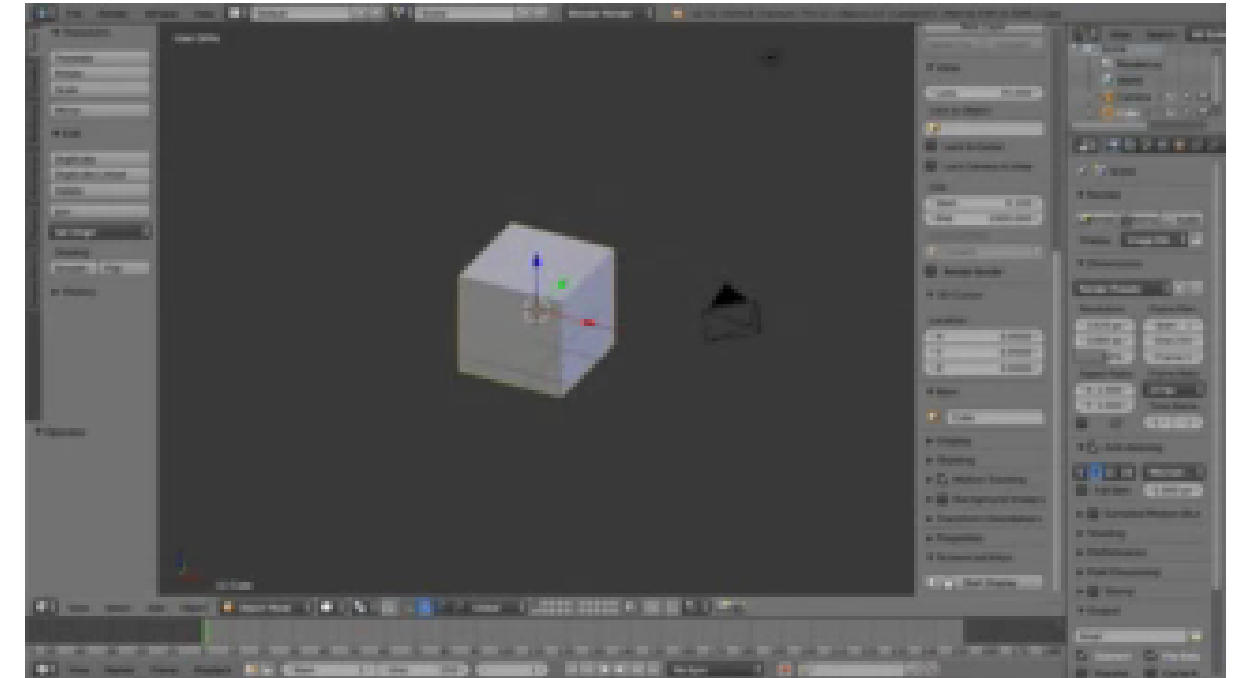
1. Low res modeling (extrusion)

2. Subdivide, Refine

- Parametric (NURBS) modeling

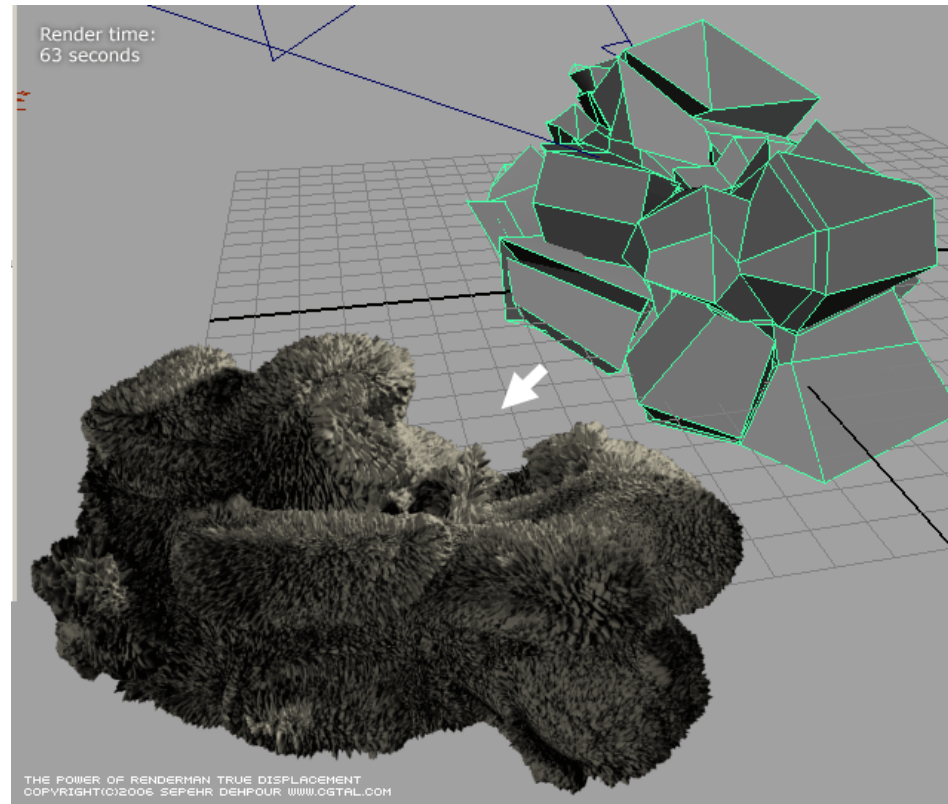
Everything else is "VFX"

Common tools: Maya, 3DSMax, Cinema4D, Blender, etc.

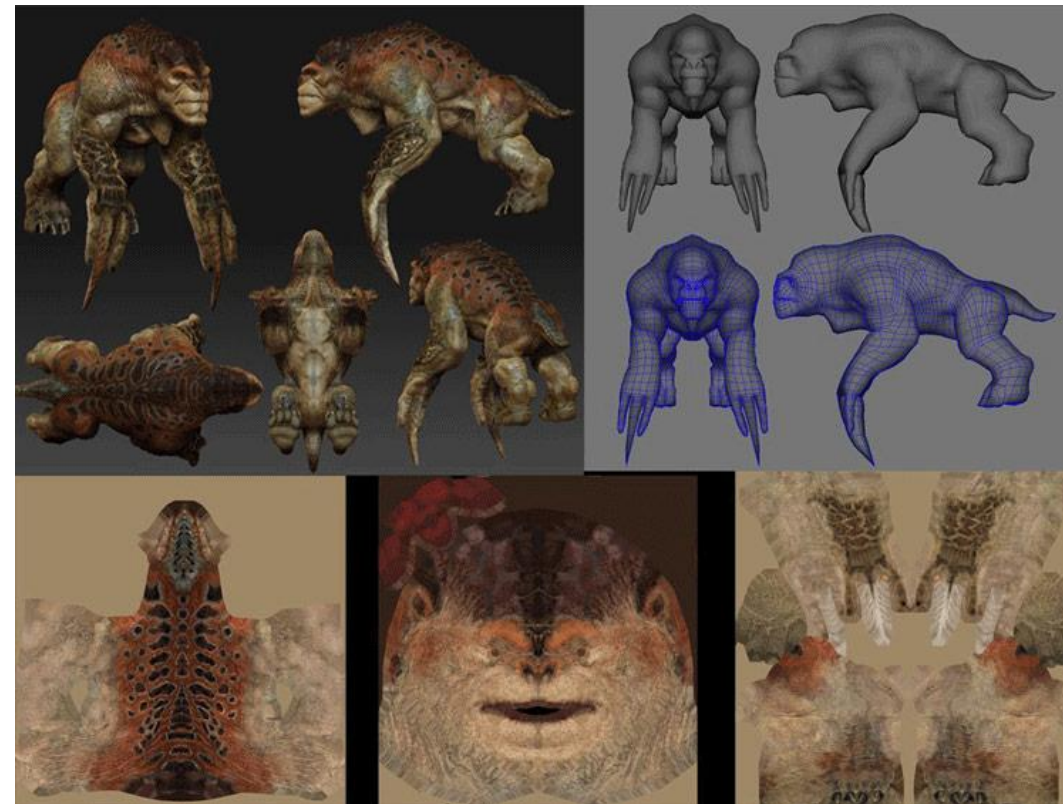


Modeling appearance - Rendering purpose

70/81



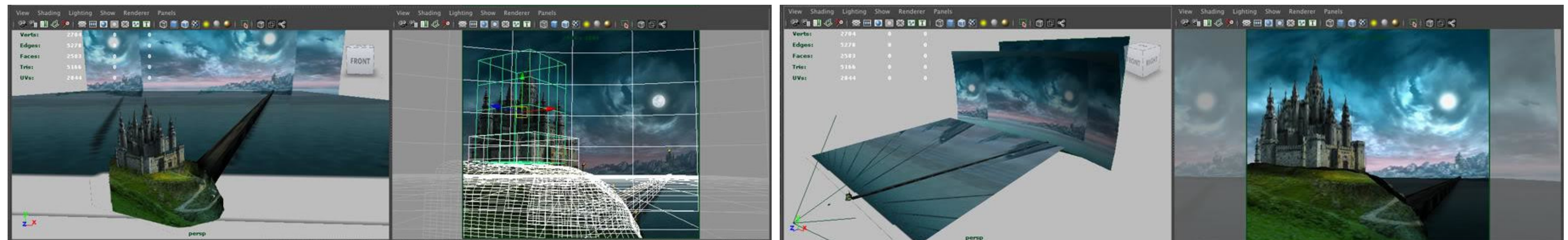
Shading



Texturing



Lighting



Mate painting

Animation

72/81

In production terminology: Animation = Key-frame animation of the rigged character
Set animation curves on rig controlers
Everything else is VFX



Animating the walk cycle (× 40)



Result

Up to 75% of artists production studios are animators

Animation = The key element - higher cost - for production studios

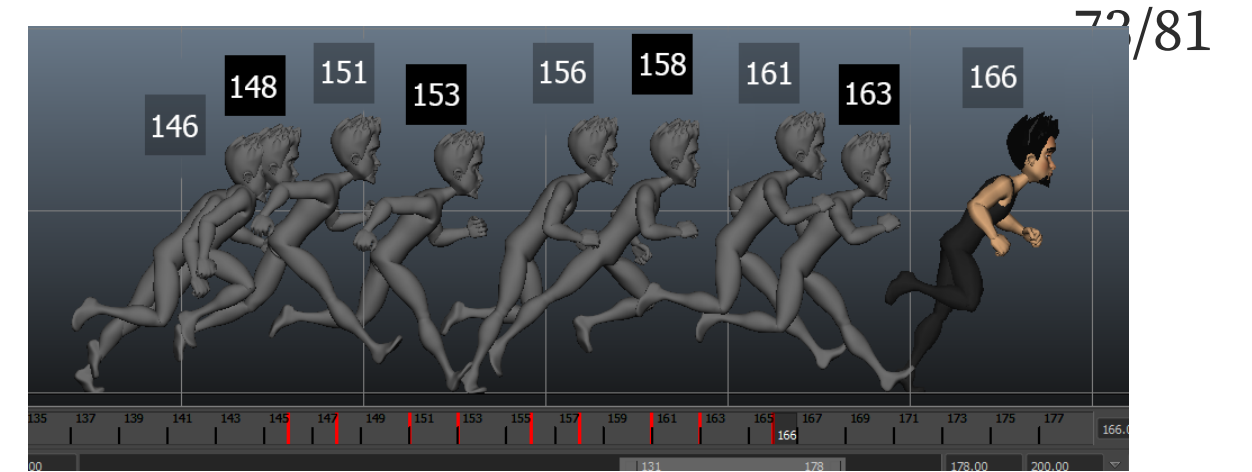
One animator → 1-10 s of animation per day

Animation sub-parts

1- **Posing** the *key frames*

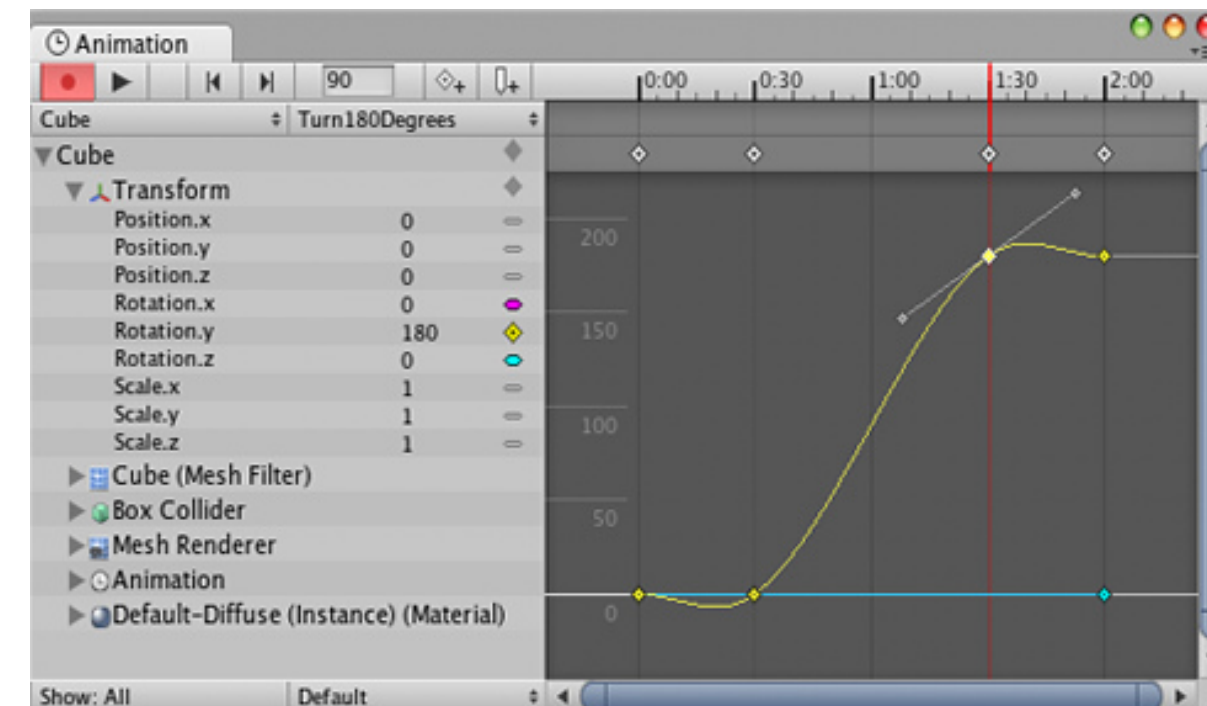
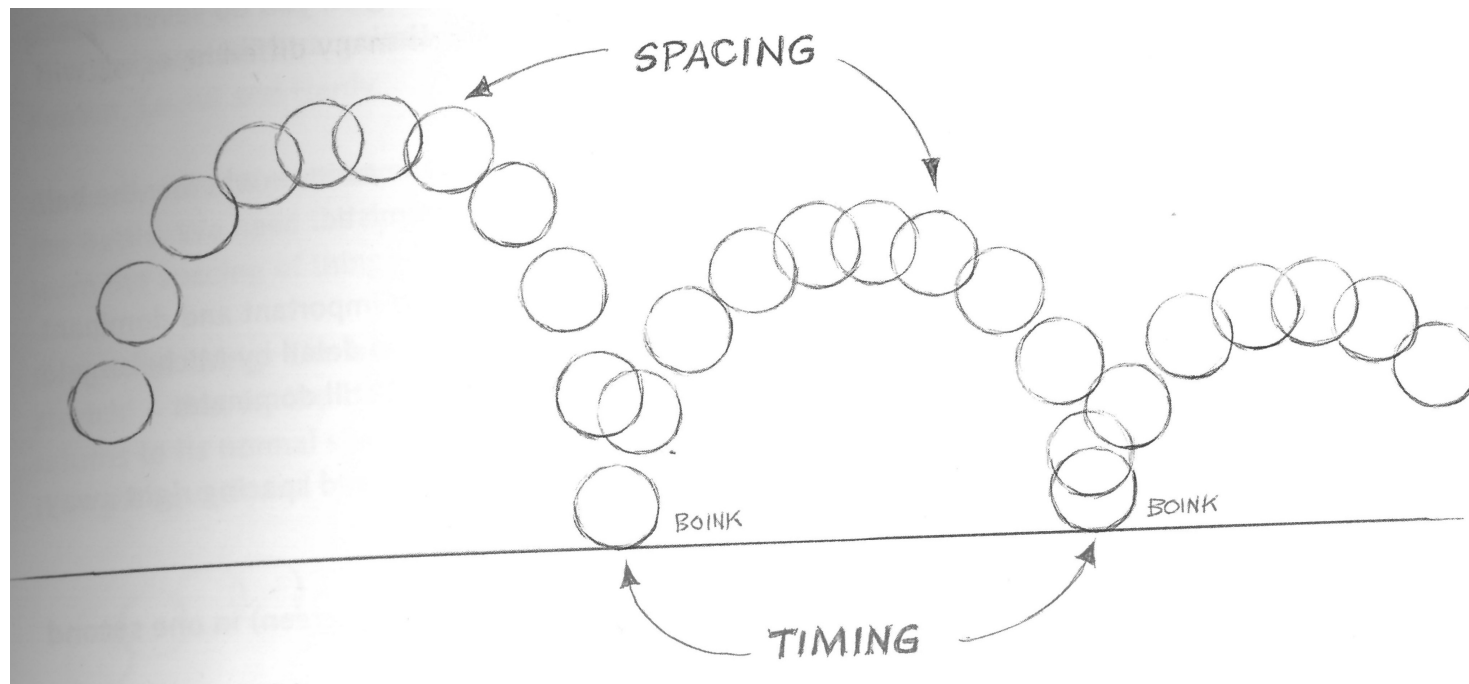
Set the main general posture of the character

*Linked to geometric **character deformation**, time is not involved*



2- **Animating** the in-betweens

- **Timing** : Place key frames at specific times (global length of an action)
- **Spacing** : Speed of the interpolation (dynamic of the action)



Special Effects (VFX)

74/81

Everything which is not handled by traditionnal modeling/rigging/animation

Physics (explosion, fluids, dynamic hairs, cloth, ...), particles systems, complex shape, crowd, etc.

Technical R&D part: One element can lead to the development of a dedicated system.

Main software: Houdini (SideFX)



Post Production

75/81

Compositing

Blend all layers: Rendered and real ones

Note: Rendering of color layers but also depth and normals.

Main software: Nuke (Foundry)



Expressive Animation

Character Animation Posing - Line of Action

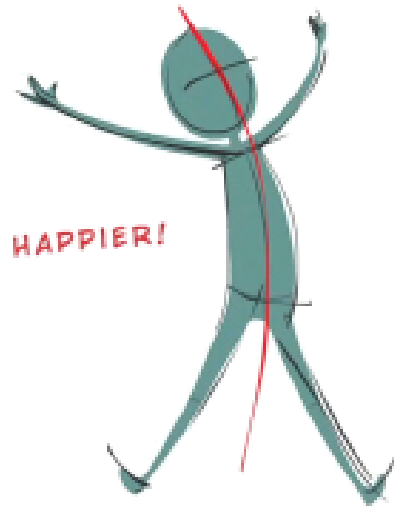
77/81

- Line of action: *Medial axis* expressing the character pose
- Express *statically* the dynamic of the action
 - Unstable pose $\Rightarrow$ Dynamic action/motion

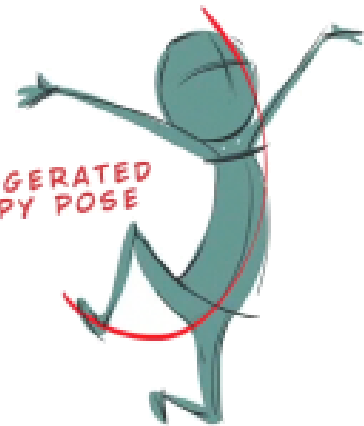
SLIGHT CURVE



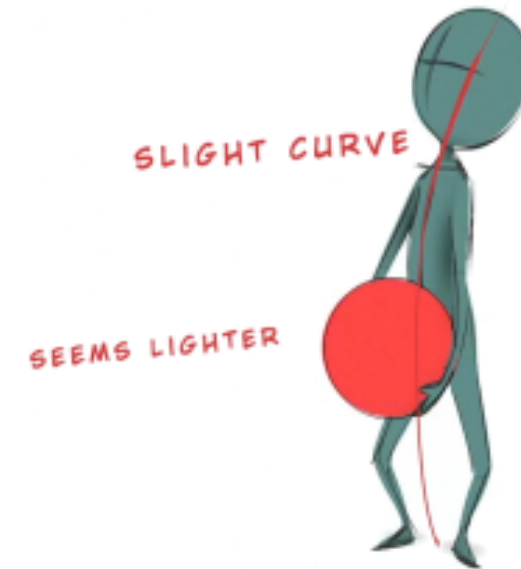
HAPPIER!



EXAGGERATED
HAPPY POSE



SEEMS LIGHTER



EXAGGERATED EFFORT POSE

SEEMS HEAVIER

ACCENTUATED
CURVE



SLIGHT CURVE



ACCENTUATED
CURVE



EXAGGERATED
SAD POSE



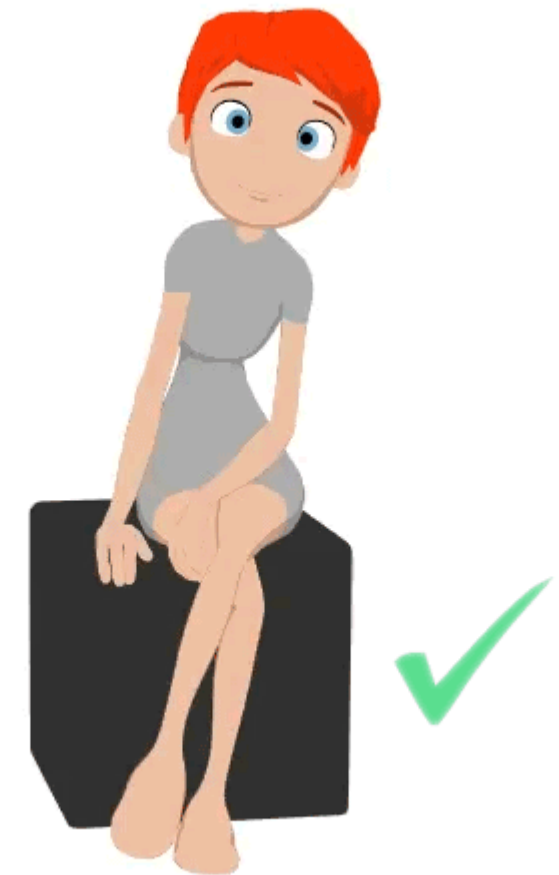
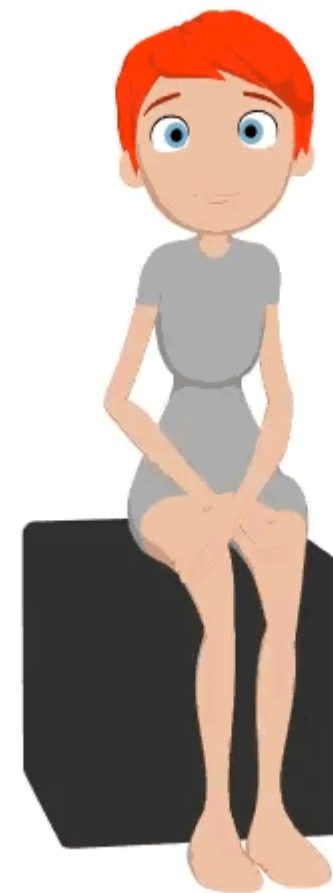
(C) Chiara Porri

Principles of animation

78/81

- Interpolation between realistic poses isn't enough for expressive animation
- *12 principles of animation* by Disney *Illusion of Life*, 1981

1. Timing
2. Spacing
3. Slow-in, Slow-out
4. Squash & Stretch
5. Anticipation
6. Follow Through
7. Secondary Action
8. Exaggeration
9. Appeal
10. Arcs
11. Staging
12. Straight Ahead/Pose to Pose



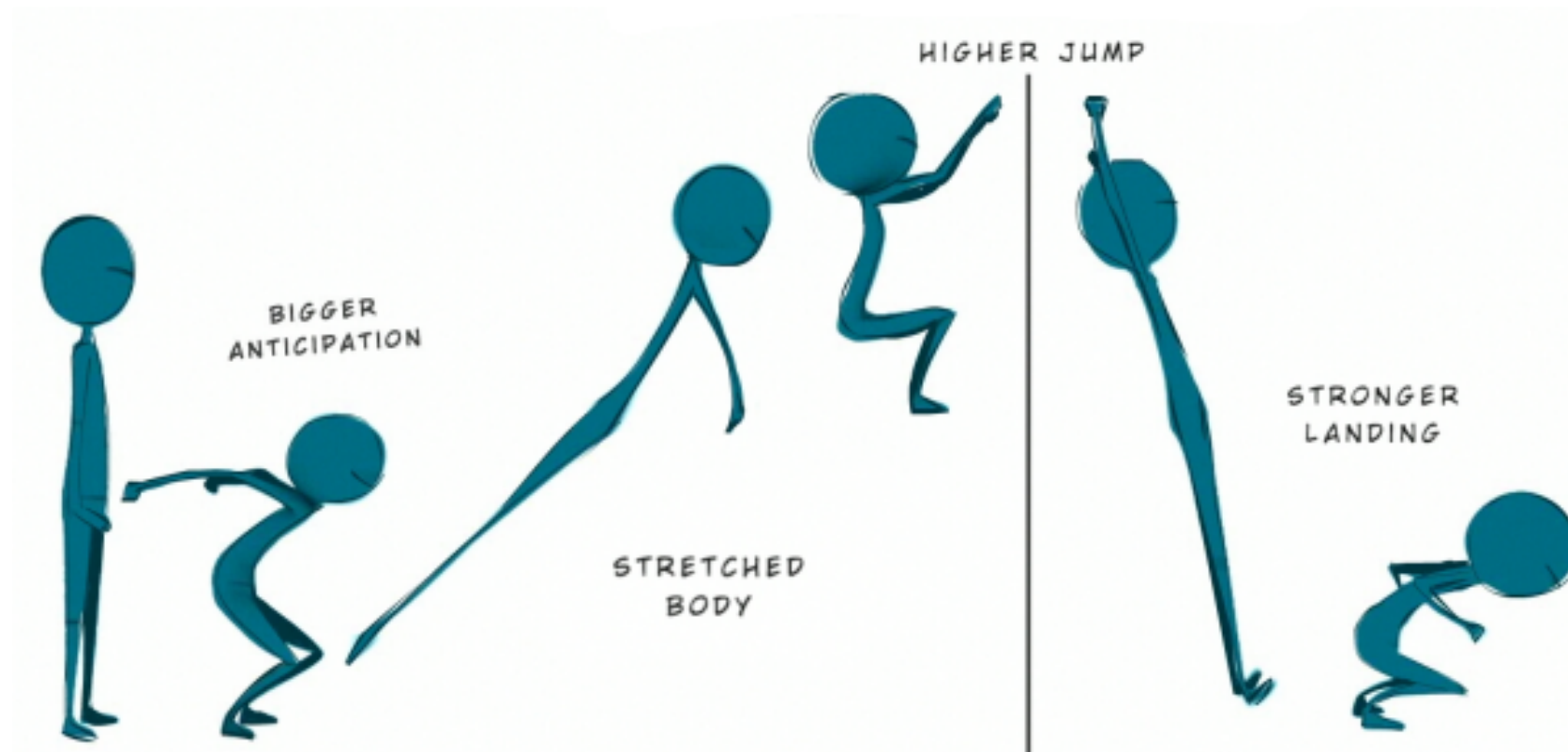
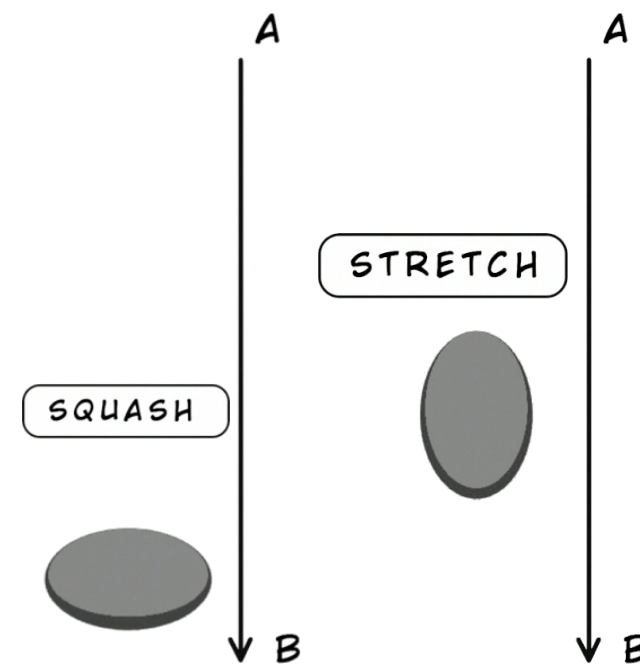
(C) Chiara Porri

Expressive animation

79/81

Squash & Stretch

- Very common in cartoon
- Unrealistic, but surprisingly *plausible*

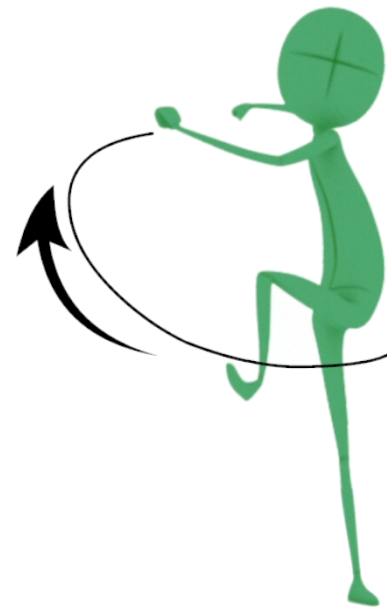


(C) Chiara Porri

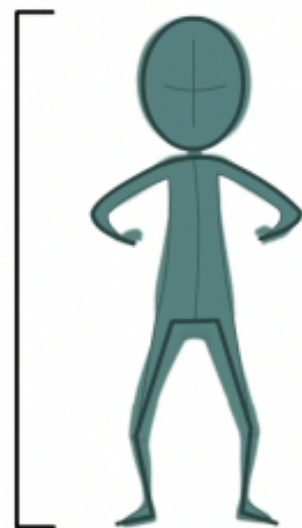
Expressive animation

80/81

Anticipation

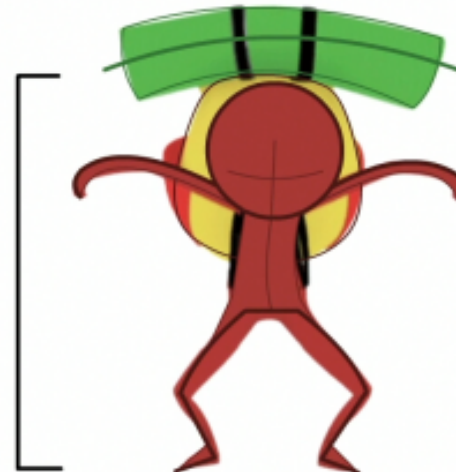


the character is pretty light



Softer Anticipation

the character is bringing
an heavy object



Stronger Anticipation

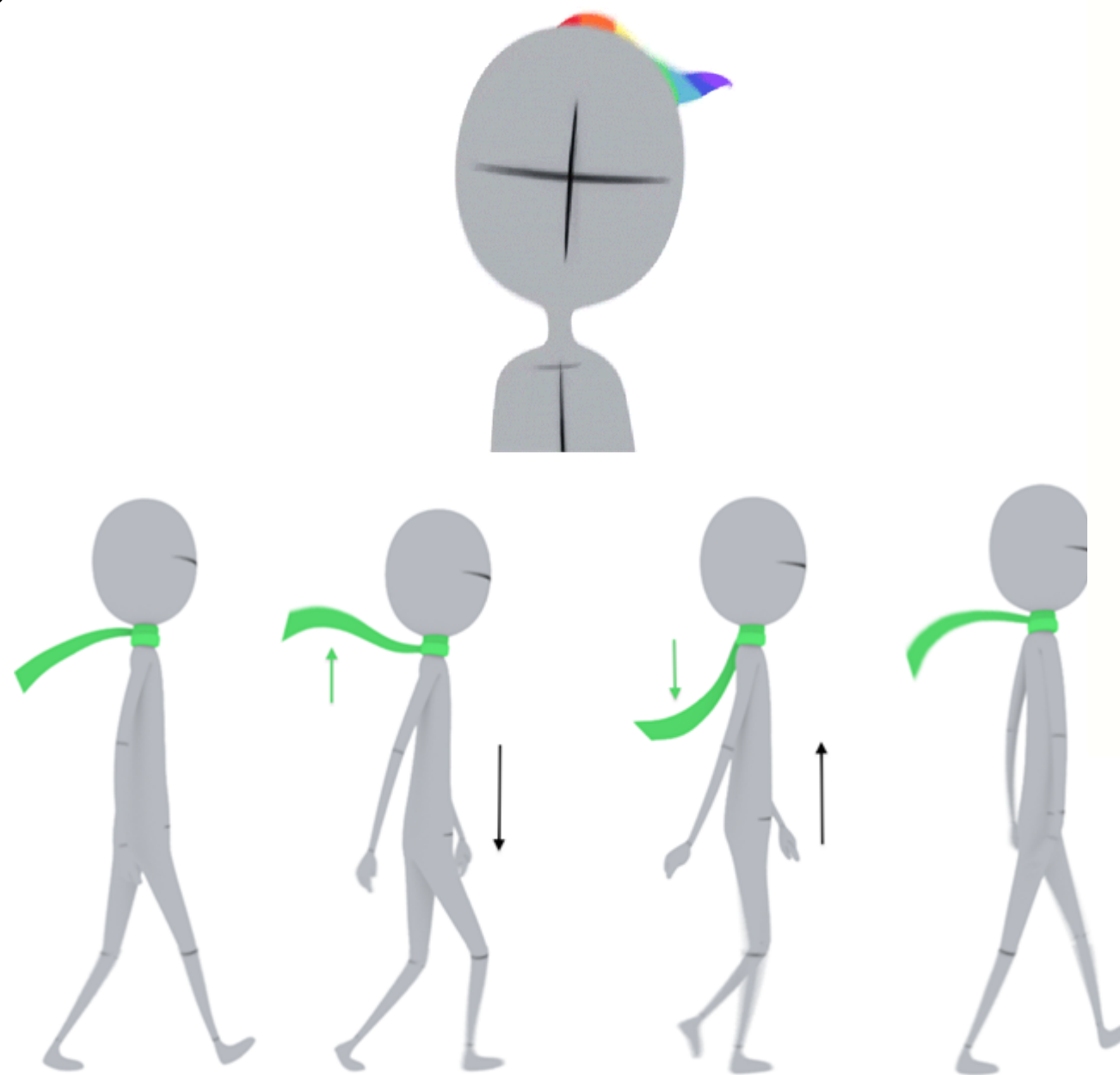
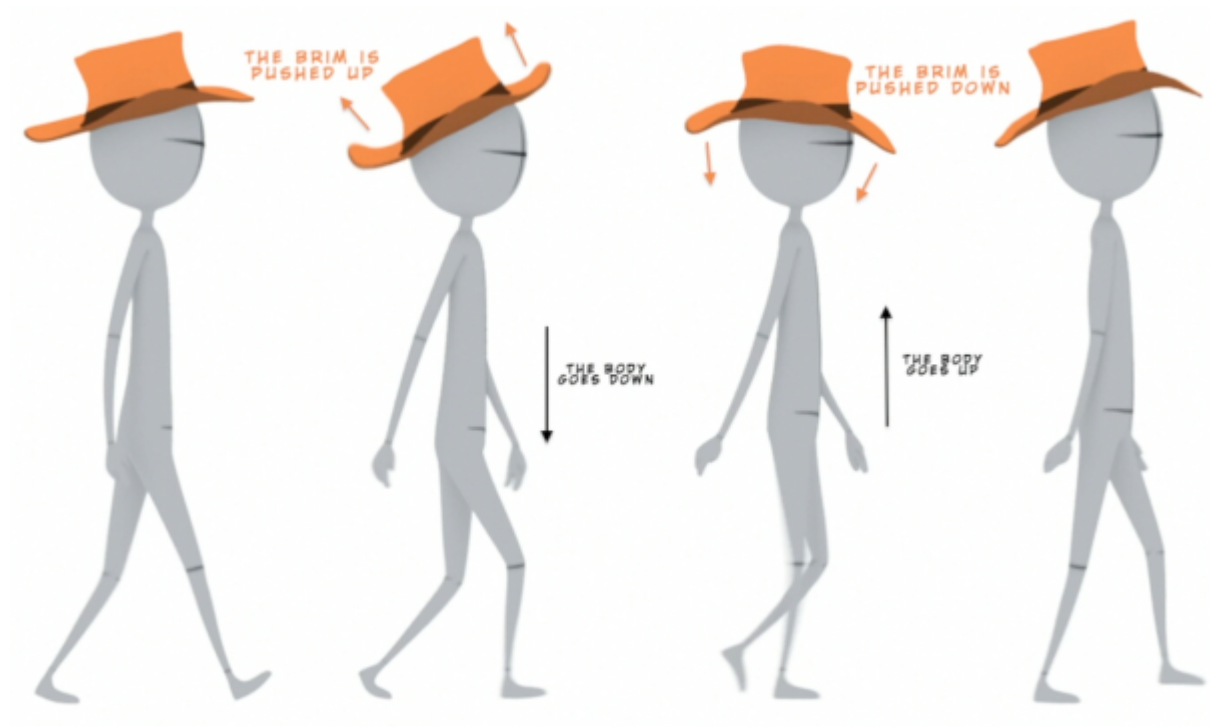


(C) Chiara Porri

Expressive animation

81/81

Follow Through / Secondary motions



(C) Chiara Porri