

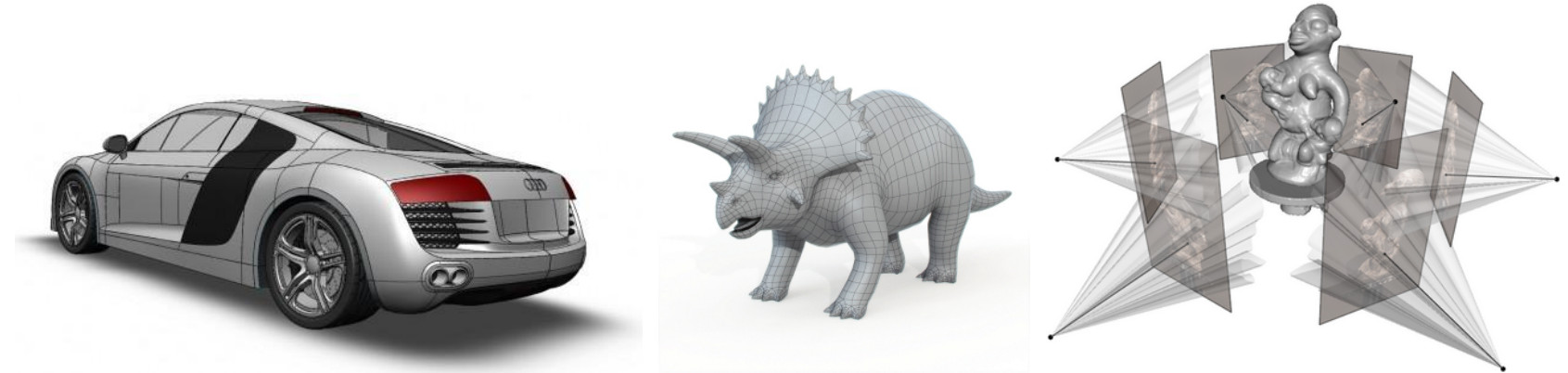
Computer Graphics notions and CG programming

Computer Graphics main SubFields

2/139

Modeling

How to create static shapes



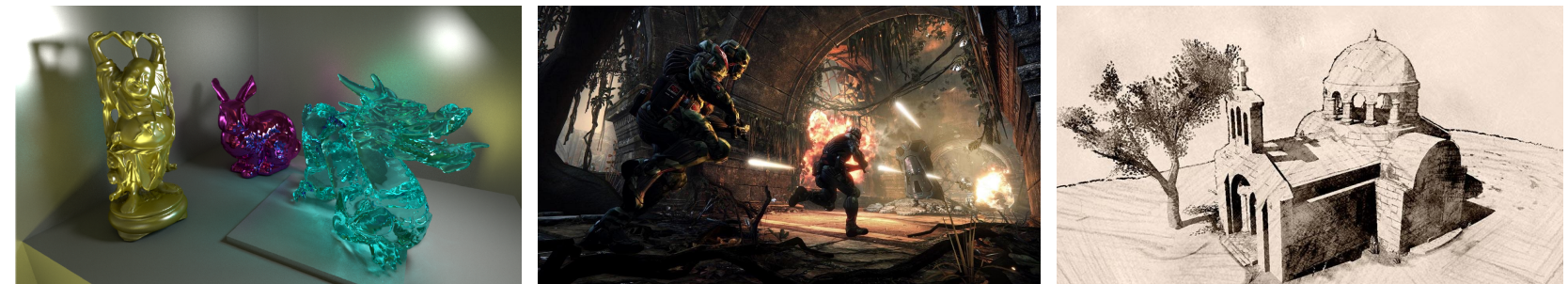
Animation

How to create and author time varying shapes



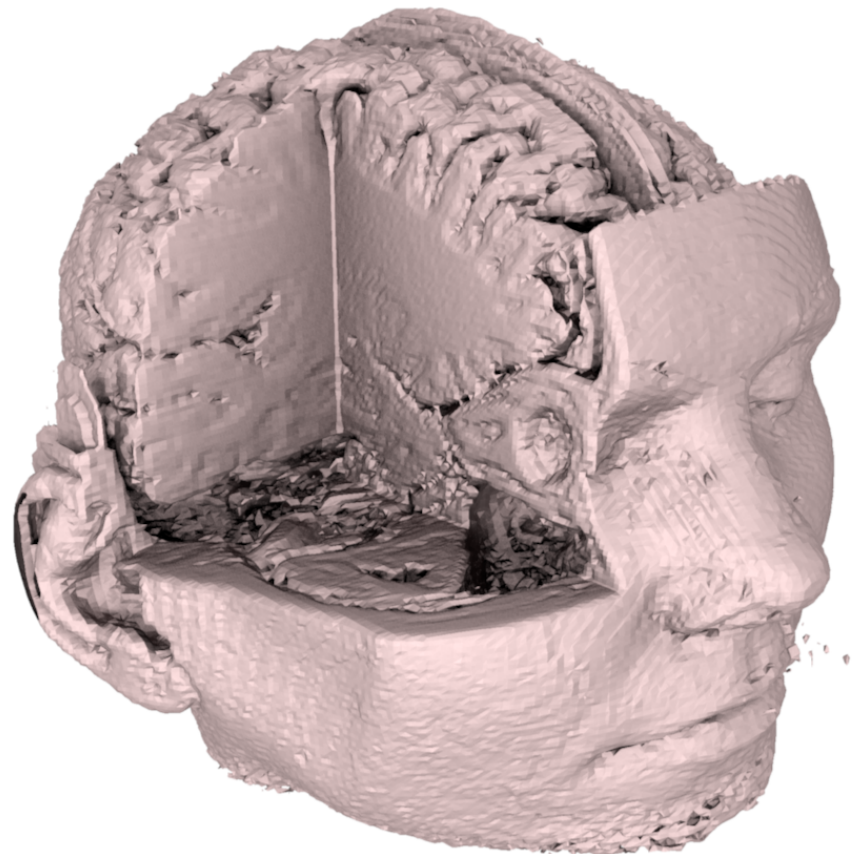
Rendering

How to generate 2D images from 3D data



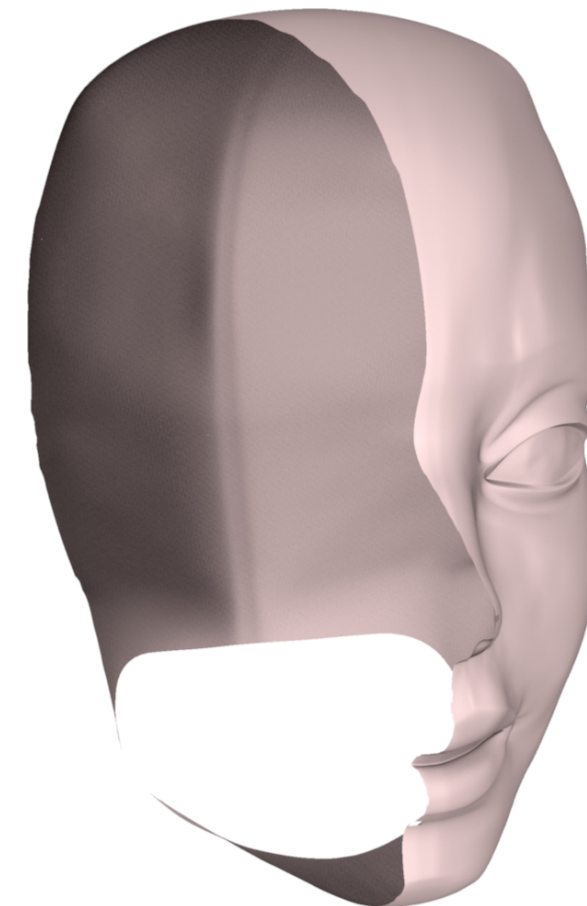
Representing 3D shapes for Graphics Applications ^{3/139}

Volume representation



+ Accurate, handle density

Surface representation



+ Focus on visible part

+ Fast GPU rendering, low memory footprint

=> **Computer Graphics:** Mostly focus on representing **Surfaces**

=> **Scientific visualization:** Volume data

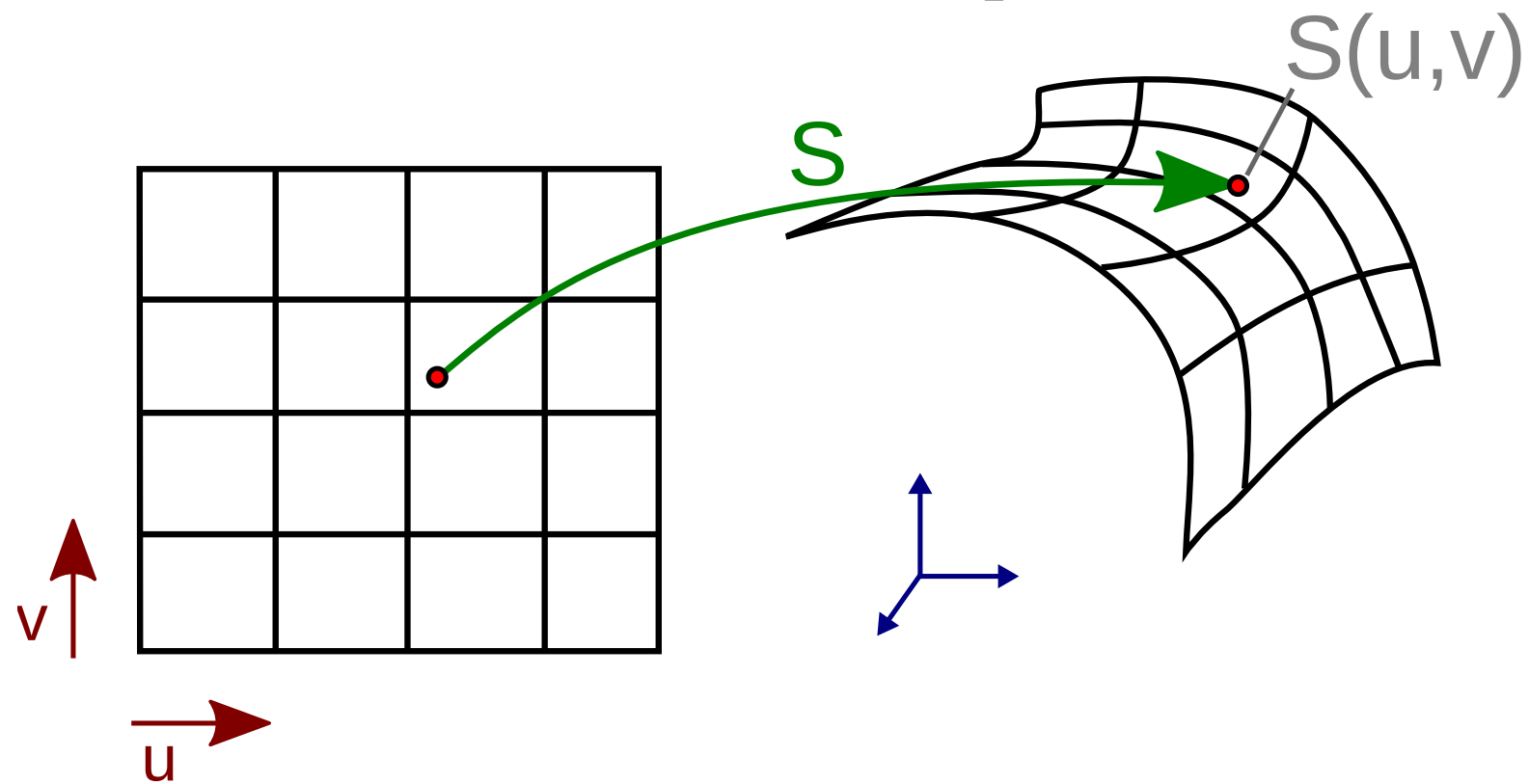
Two main representations for surfaces

4/139

Explicit representation

$$S(u, v) = (x(u, v), y(u, v), z(u, v))$$

Parametric map

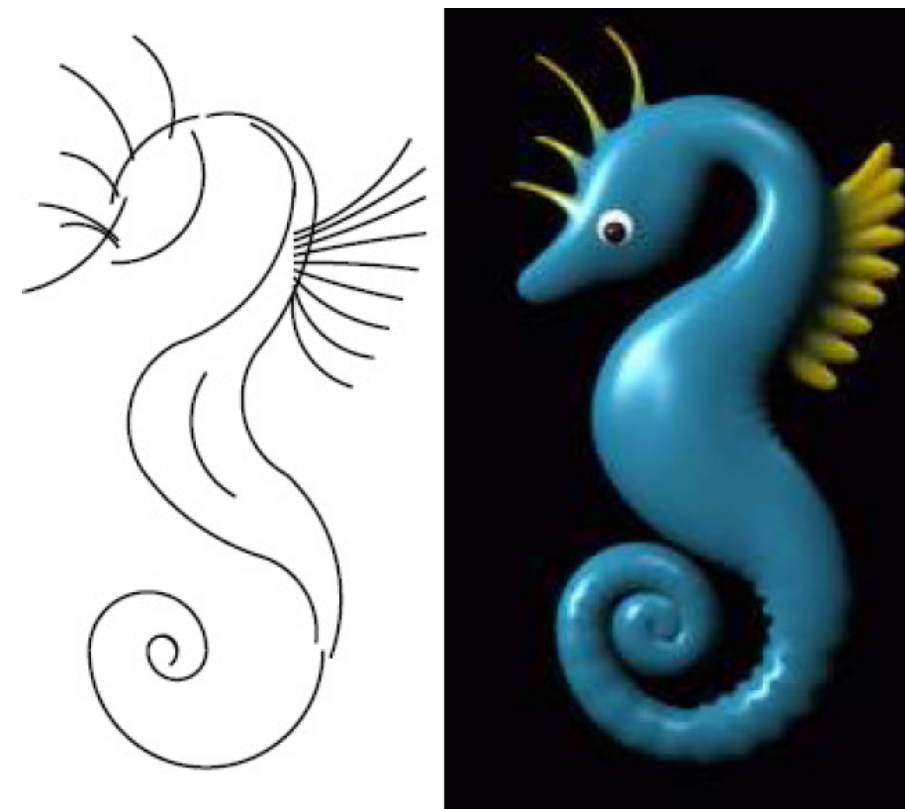


+ Neighborhood information

Implicit representation

$$S = \{(x, y, z) \in \mathbb{R}^3 \mid F(x, y, z) = 0\}$$

Isosurface of scalar field



+ Topological modification

Two main representations for surfaces

5/139

Example for a sphere

Explicit representation

$$S(u, v) = (x(u, v), y(u, v), z(u, v))$$

Parametric map

$$S(u, v) = \begin{cases} x(u, v) = R \sin(u) \cos(v) \\ y(u, v) = R \sin(u) \sin(v) \\ z(u, v) = R \cos(u) \end{cases}$$

Implicit representation

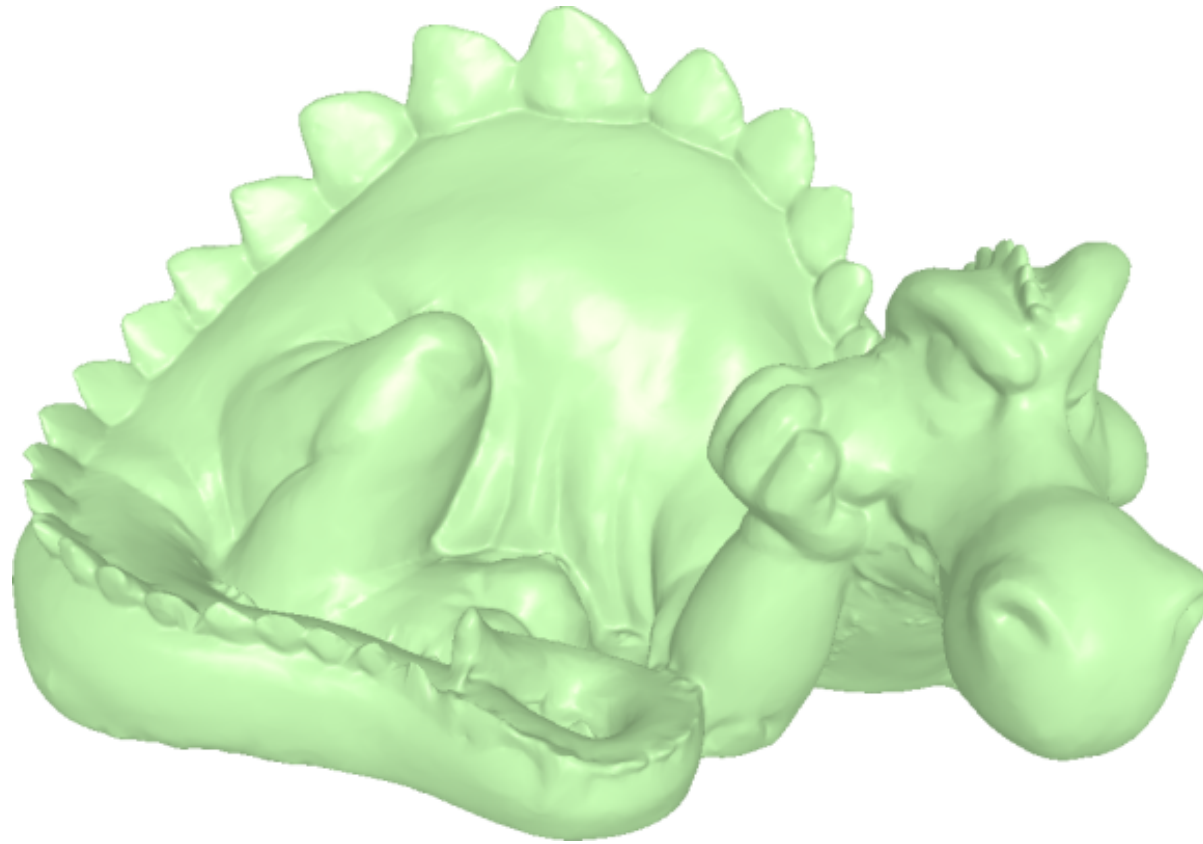
$$S = \{(x, y, z) \in \mathbb{R}^3 \mid F(x, y, z) = 0\}$$

Isosurface of scalar field

$$F(x, y, z) = x^2 + y^2 + z^2 - R^2$$

Difficulty of surface representation using functions ^{6/139}

Which function can represent this shape ?



$$S(u, v) = ?$$

$$F(x, y, z) = ?$$

Objective of surface representation

7/139

Main Idea => Use of **piecewise approximation**

Ideal surface representation

- **Approximate** well any surface
- Require **few samples**
- Can be **rendered** efficiently (GPU)
- Can be manipulated for **modeling**

Example of models:

- *Mesh-based: Triangular meshes, Polygonal meshes, Subdivision surfaces*
- *Polynomial: Bezier, Spline, NURBS*
- *Implicit: Grid, Skeleton based, RBF, MLS*
- *Point sets*

=> For projective/rasterization render pipeline : always render **triangular meshes** at the end

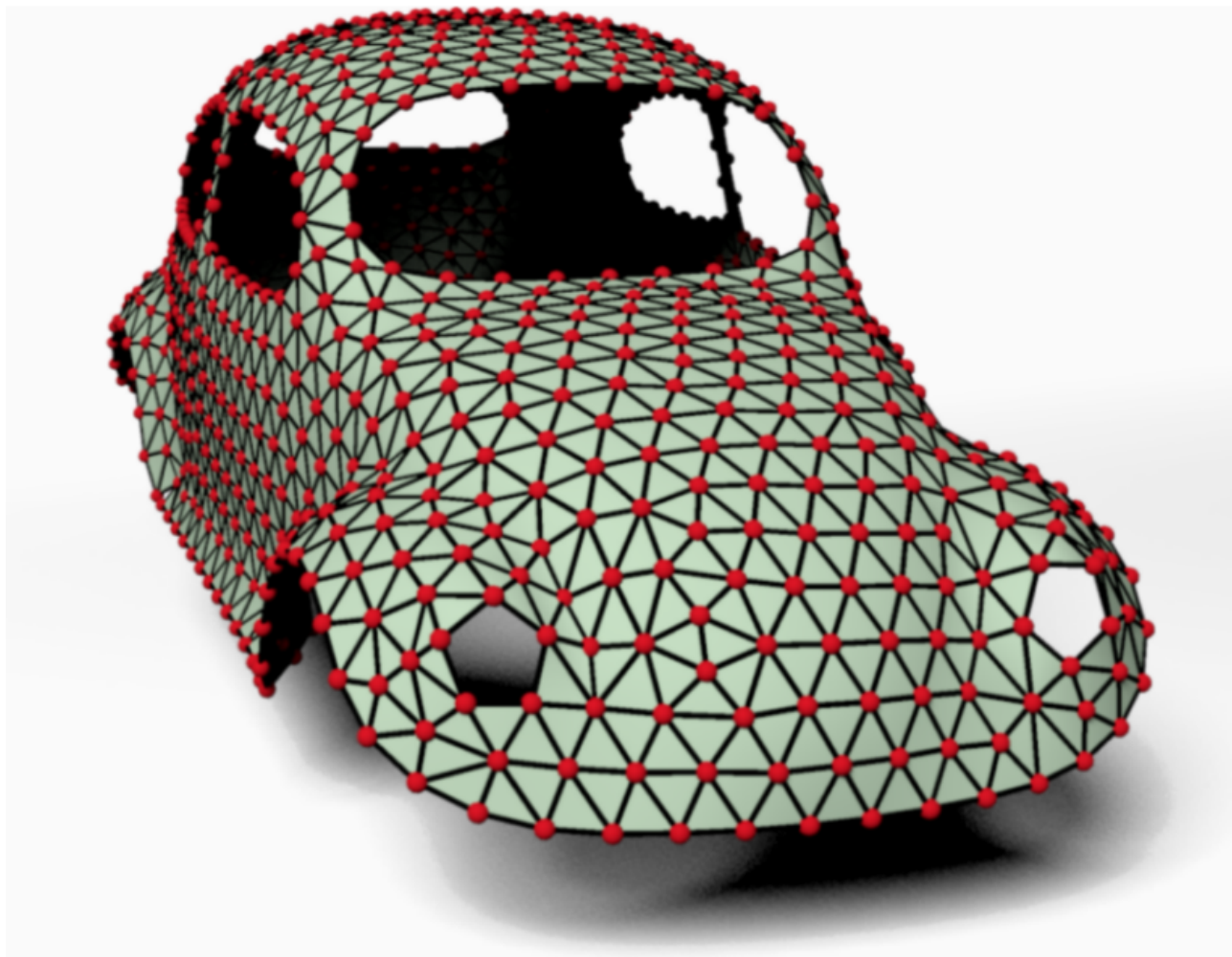
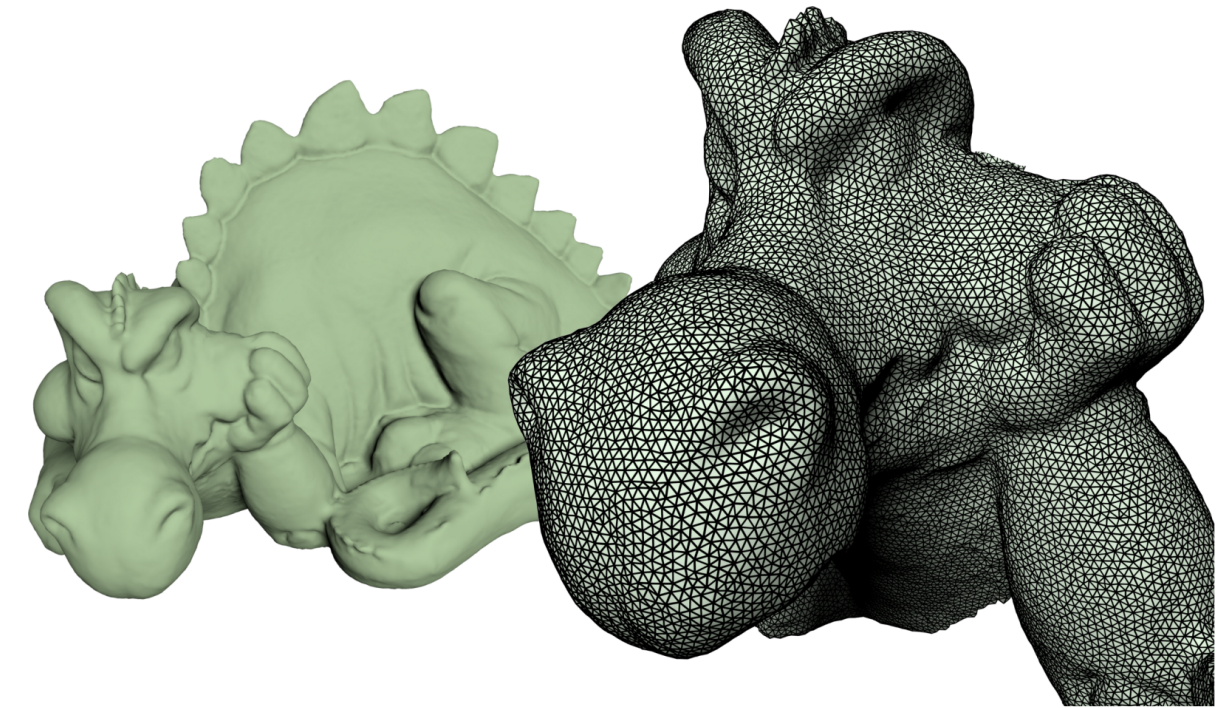
- + Simplest representation
- + Fit to GPU Graphics render pipeline
- Requires large number of samples: complex modeling
- Tangential discontinuities at edges

Meshes

Simplest possible representation of 3D surfaces: set of triangles

Described as a triplet: (Vertices, Edges, Faces)

$$S = (\mathcal{V}, \mathcal{E}, \mathcal{F})$$



● Vertex

$$\mathcal{V} = (v_1, \dots, v_N)$$

／ Edge

$$\mathcal{V} = (v_1, \dots, v_{N_e}) \in (\mathcal{V}^2)^{N_e}$$

▲ Face

$$\mathcal{F} = (f_1, \dots, f_N) \in (\mathcal{V}^3)^{N_f}$$

Mesh encoding

9/139

Exemple for a tetrahedron

- 1st Solution: *Soup of polygons*

```
triangles = [(0.0, 0.0, 0.0), (1.0, 0.0, 0.0), (0.0, 0.0, 1.0),  
             (0.0, 0.0, 0.0), (0.0, 0.0, 1.0), (0.0, 1.0, 0.0),  
             (0.0, 0.0, 0.0), (0.0, 1.0, 0.0), (1.0, 0.0, 0.0),  
             (1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)]
```

- 2nd solution: *Geometry, Connectivity*

```
geometry = [(0.0, 0.0, 0.0), (1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)]  
connectivity = [(0,1,3), (0,3,2), (0,2,1), (1,2,3)]
```

=> Preferred solution

+ more space efficient

+ modifying 1 vertex = 1 operation

Mesh buffer encoding in C++

10/139

```
#include <vector>
#include <array>

struct vec3 {float x,y,z;};
using index3 = std::array<unsigned int, 3>;

int main()
{
    std::vector<vec3> geometry = { {0.0f, 0.0f, 0.0f}, {1.0f, 0.0f, 0.0f},
                                   {0.0f, 1.0f, 0.0f}, {0.0f, 0.0f, 1.0f} };
    std::vector<index3> connectivity = { {0,1,3}, {0,3,2}, {0,2,1}, {1,2,3} };

    return 0;
}
```

Example of 3D Mesh file

11/139

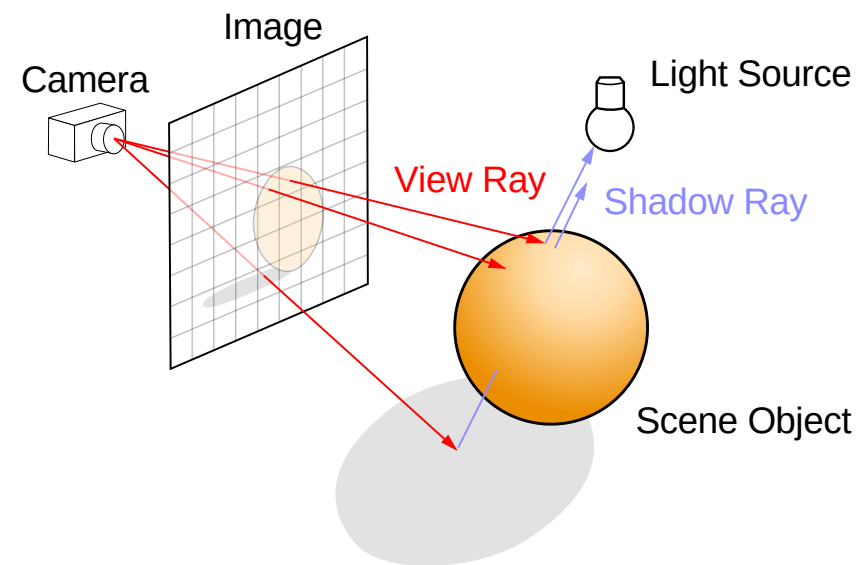
```
v -0.000000 0.620276 0.108446
v -0.000000 0.685780 0.104094
v 0.011128 0.685780 0.102245
v 0.014793 0.620276 0.106125
v 0.034724 0.684975 0.079817
v 0.040413 0.620278 0.086828
v 0.029160 0.619800 0.099405
v 0.024110 0.685194 0.093530
v 0.046714 0.554312 0.085764
v 0.033793 0.547222 0.100284
v 0.015067 0.542780 0.113608
v -0.000000 0.541146 0.117759
v 0.051177 0.430214 -0.047903
v 0.049948 0.435812 -0.035967
v 0.028863 0.449897 -0.050037
v 0.028839 0.444346 -0.059194
v 0.017691 0.251925 0.023686
v 0.034131 0.252216 0.014535
v 0.036689 0.275442 0.012672
v 0.015166 0.271140 0.025837
v 0.014000 0.205441 0.024057
```

Open question: How to display it efficiently on screen?

How to render surfaces

12/139

Ray tracing

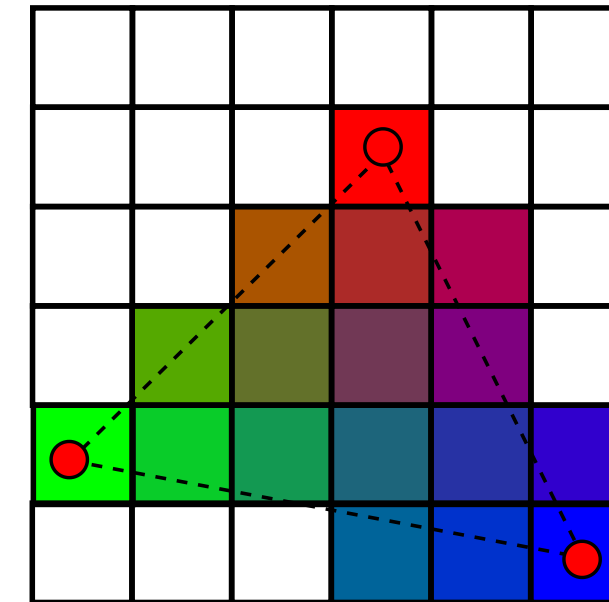


- Throw rays from light-sources/camera
- Intersect rays with 3D shapes
- Pixel-wise computation

- + Photo-realistic rendering
(Soft shadows, reflection, caustics)
- + Handle general surfaces
- High computational cost

=> Restricted to offline rendering (but developing more and more)

Projection/Rasterization



- Assume shapes made of triangles
 1. Project each triangle onto camera screen space
 2. Rasterize projected triangle into pixels
- Triangle-wise computation

- + Efficiently implemented on GPU
- Limited to triangles
- No native effects (shadows, transparency, etc)

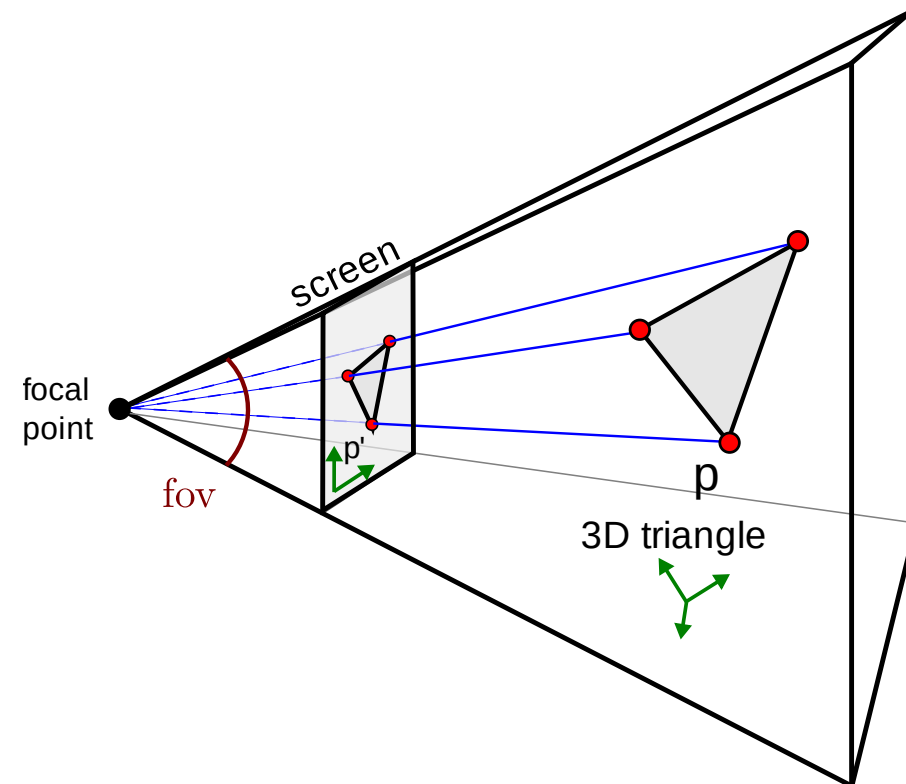
=> The standard real time rendering with GPU

Projection/Rasterization

13/139

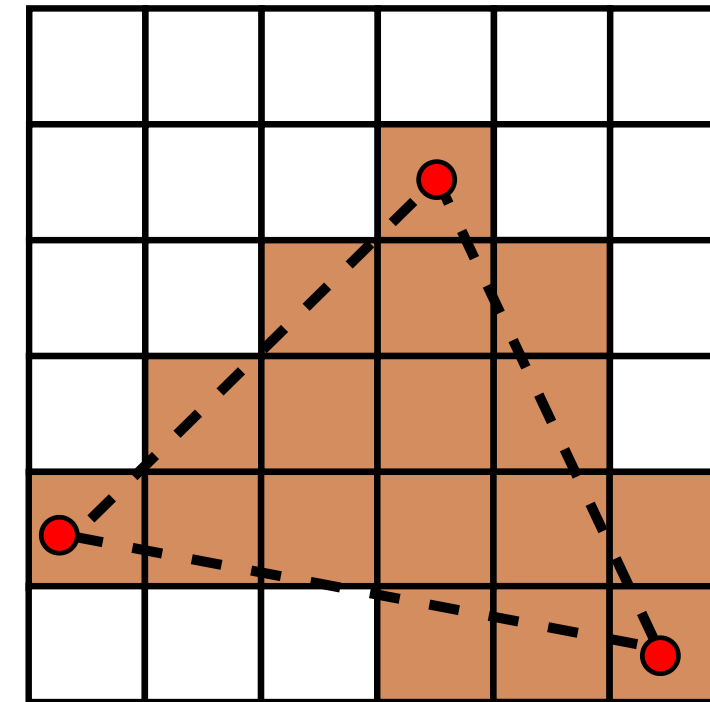
Object made of **triangles only**

1. Project vertices of triangles



- Projection computed as matrix operation
(projective space $p' = M p$)

2. Rasterization



- Discrete geometry
- Interpolate attributes (colors, etc) on each pixel

=> At interactive frame rate (≥ 25 fps)

- Project all triangles of shapes
- Fill all pixels of each projected triangle

Quick fundamental notions for practical 3D programming

- Affine transform as 4D matrices
- Perspective and projective space
- Illumination and normals

Affine transforms and 4D vectors/matrices

15/139

Preliminary note

- We use a lot affine transformations to place shapes in 3D space

Translation, Rotation, Scaling

- In CG vectors are often expressed in 4D, and matrices are 4×4 .

=> Reason: Affine transforms can be expressed linearly (with matrices) in 4D

$$p = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad M = \begin{pmatrix} m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Affine transform in 2D

16/139

General principle in the 2D case

Example for a point $p = (x, y)$

$$\text{Rotation } \mathbf{R} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}, \quad \text{Scaling } \mathbf{S} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}, \quad \text{Translation } (x + t_x, y + t_y) \text{ (not linear)}$$

Cannot express conveniently composition b/w several rotation, scaling, translation.

Trick - Add an extra coordinates to points $p = (x, y, 1)$ (homogeneous coordinates).

$$\text{Then translation can be expressed linearly } p' = \mathbf{T} p, \text{ with } p' = \underbrace{\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

$$\text{Similarly with rotation } \mathbf{R} = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \text{and scaling } \mathbf{S} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Affine transform matrix

17/139

With the extra dimension (in 2D):

Translation T , rotation R , scaling S can be composed as matrix products representation

$$\text{ex. } M = T_0 R_0 S_0 T_1 R_1 S_1 \dots \quad M = \left(\begin{array}{cc|c} m_{00} & m_{01} & t_x \\ m_{10} & m_{11} & t_y \\ \hline 0 & 0 & 1 \end{array} \right).$$

m_{ij} : linear part (rotation and scaling); $t_{x/y}$: translation part

Similar in **3D** but with **4-components vectors**, and **4×4 matrices**.

$p = (x, y, z, 1)$ - represents 3D position

$$M = \left(\begin{array}{ccc|c} m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ \hline 0 & 0 & 0 & 1 \end{array} \right) - \text{represents 3D affine transformation (rotation, scaling, translation)}$$

Note: vectors and points can be expressed

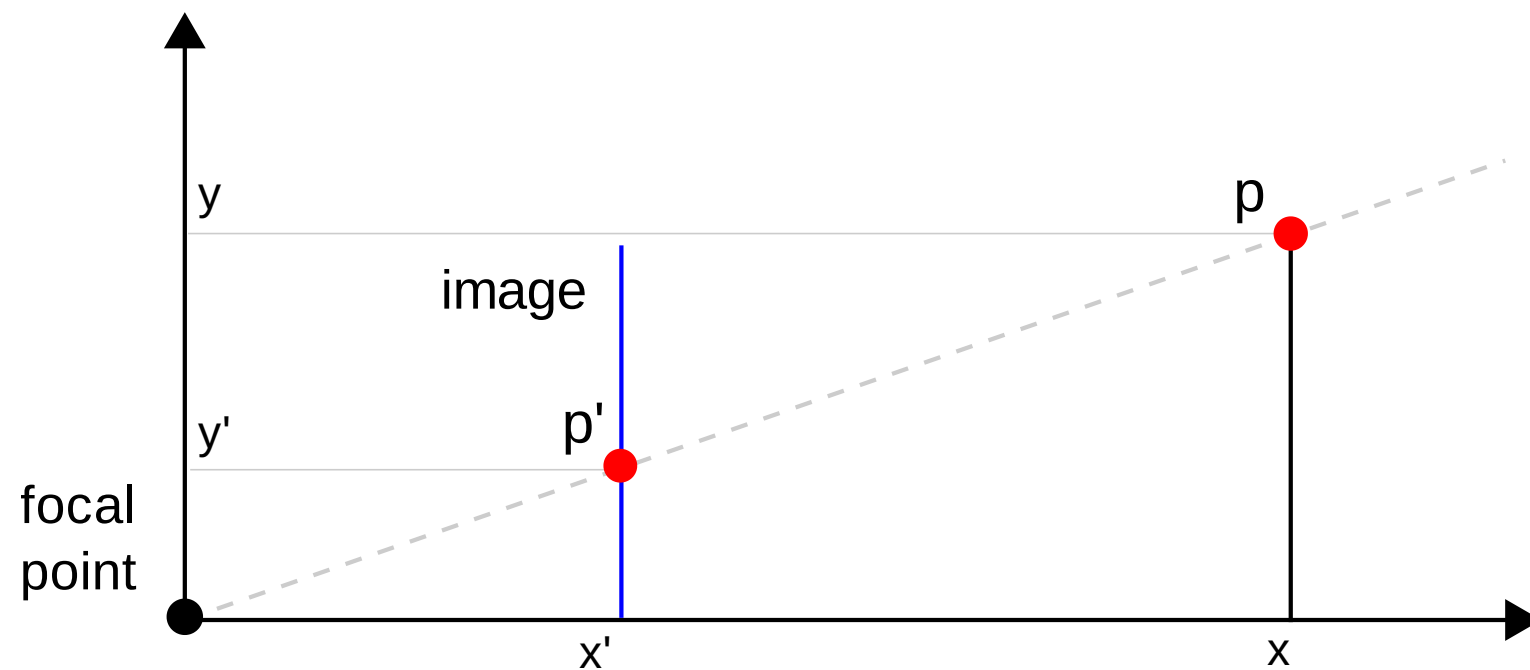
- 3D point $(x, y, z, 1)$ - translation applies.
- 3D vector $(x, y, z, 0)$ - translation doesn't apply.

Perspective and projective space

18/139

Modeling perspective projection requires division.

ex. in 2D (1D projection)



Projective space

- Real points lie on $z = 1$
- Vectors lie on $z = 0$

Real coordinates of points are obtained after normalization (division by z).

$$y' = x' \frac{y}{x} = f \frac{y}{x} \quad (f: \text{focal})$$

Linear model using 3D vectors in projective space.

$$p' = \begin{pmatrix} f \\ f \frac{y}{x} \\ 1 \end{pmatrix} \underbrace{=}_{\text{normalization}} \begin{pmatrix} fx \\ fy \\ x \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

considering that the last coordinate must always be normalized to 1 (for points).

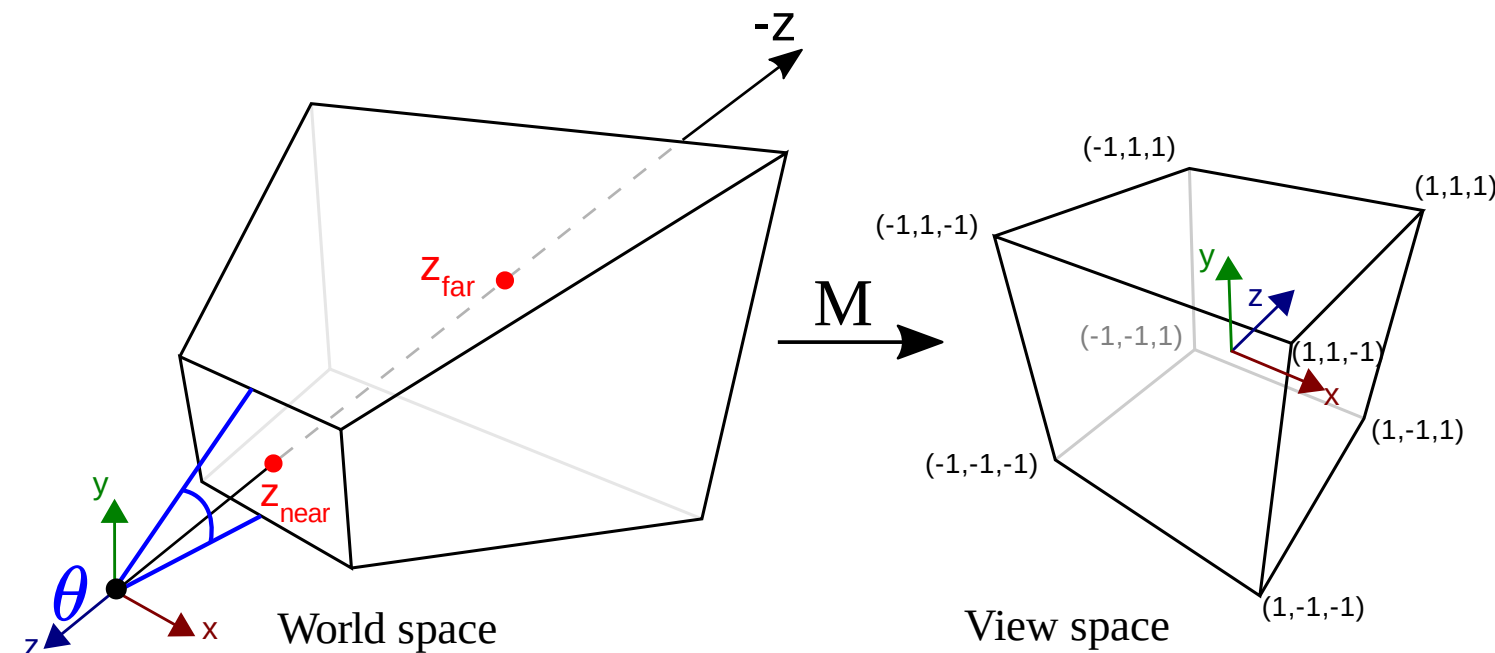
Perspective matrix

19/139

Perspective space : Allows perspective projection expressed as matrix.

Common constraints (in OpenGL)

- Wrap the viewing volume (truncated cone with rectangular basis called *frustum*) $(z_{near}, z_{far}, \theta)$ to a cube.
- θ : view angle
- $p = (x, y, z, 1) \in \text{frustum} \Rightarrow p' = (x', y', z', 1) \in [-1, 1]^3$.



$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{pmatrix} \quad \begin{aligned} f &= 1 / \tan(\theta/2) \\ L &= z_{near} - z_{far} \\ C &= (z_{far} + z_{near}) / L \\ D &= 2 z_{far} z_{near} / L \end{aligned}$$

In practice

=> You must define z_{near} , z_{far}

=> $z_{far} - z_{near}$ should be as small as possible for maximum depth precision.

To which view space coordinates are mapped 3D world space points at z_{near} , z_{far} ?

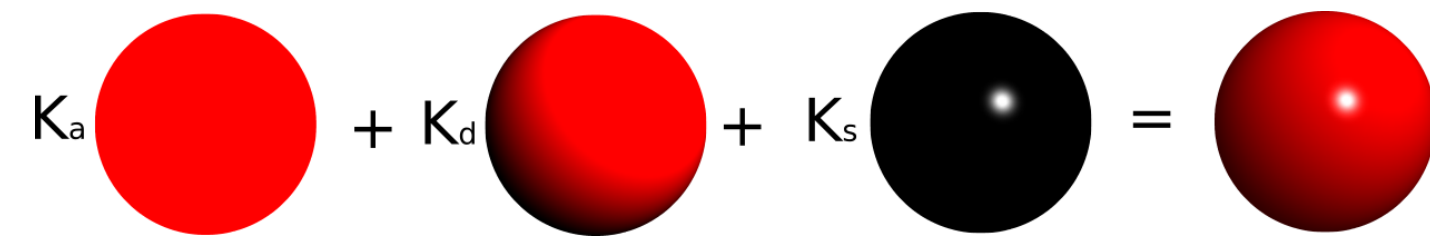
Per-vertex normal and illumination

20/139

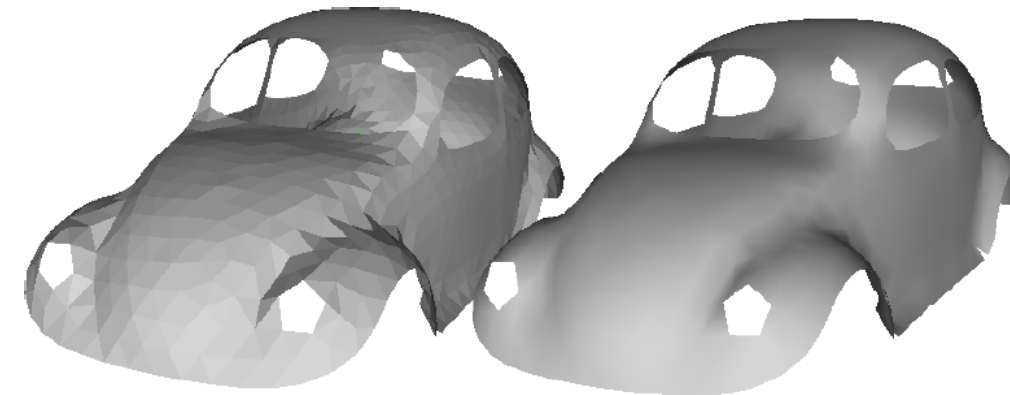
For smooth looking meshes, we define a **normal per-vertex**.

- Vertices are seen as samples on a smooth underlying surface

- Normals are used for illumination
ambient, diffuse, specular components

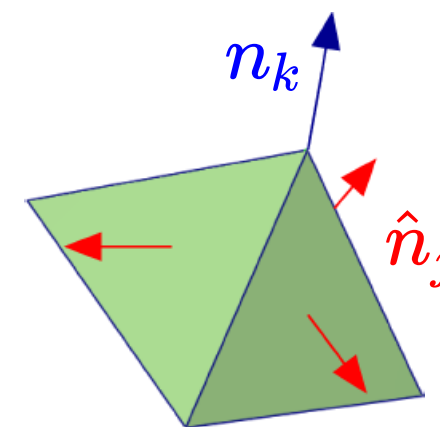


- *Phong shading* interpolates normals on each *fragments* of triangles, and compute illumination.



Possible automatic computation of normals: averaged normals of surrounding triangle.

$$n_k = \frac{\sum_{j \in \mathcal{V}_k} \hat{n}_j}{\left\| \sum_{j \in \mathcal{V}_k} \hat{n}_j \right\|}, \mathcal{V}_k: \text{neighboring triangles.}$$



Procedural Animation

Particles Trajectories

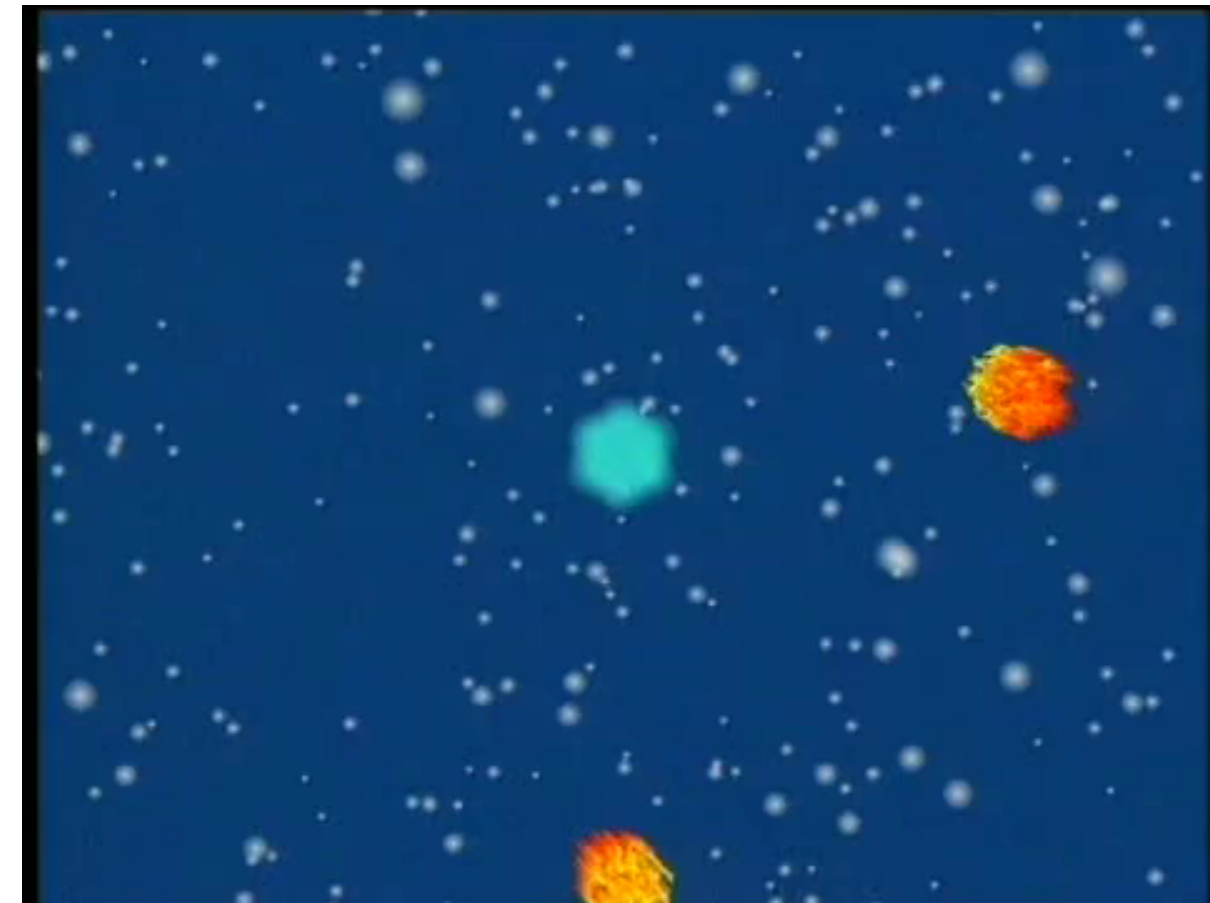
Particles Systems History

22/139

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

Free fall of spheres under gravity

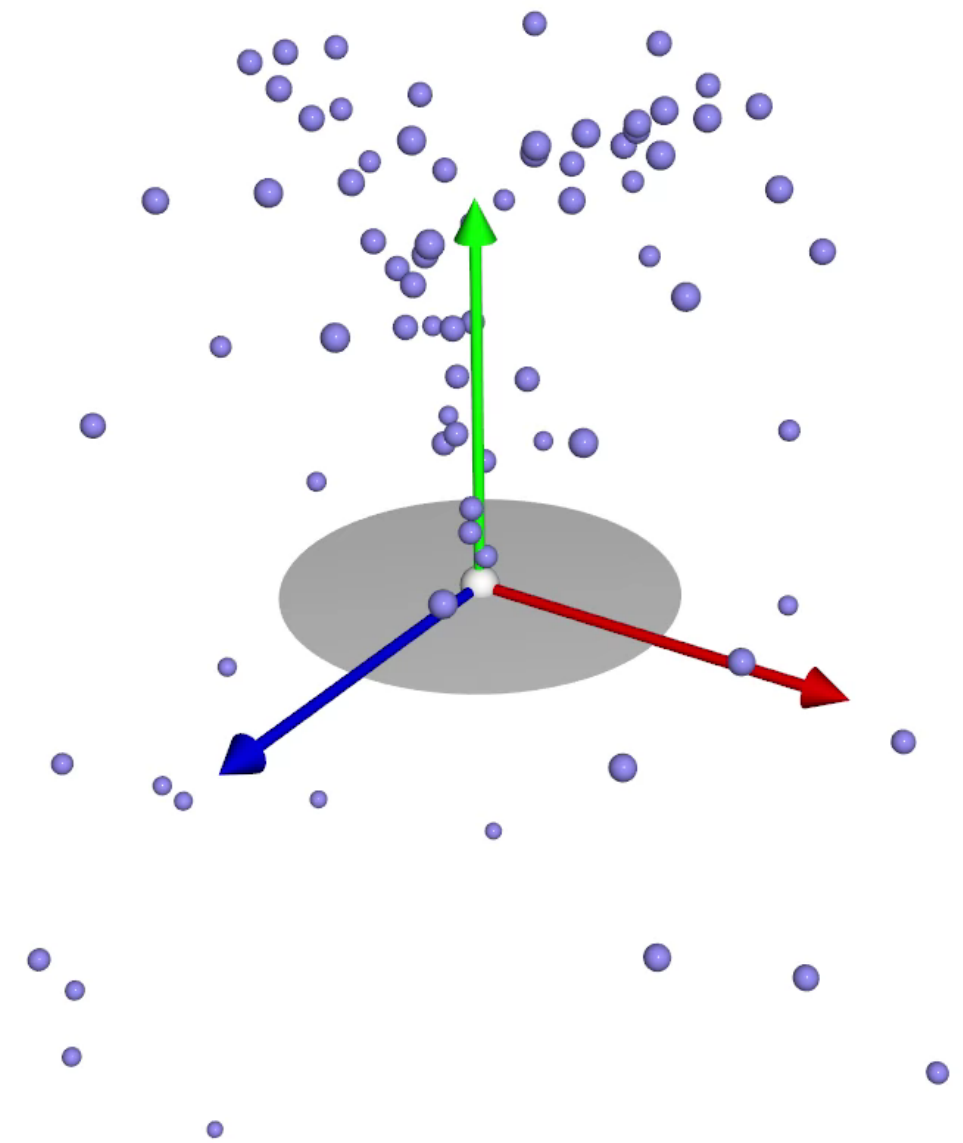
- Geometrical representation of each particle: sphere
- Equation of motion $p(t) = \frac{1}{2}gt^2 + v_0t + p_0$
- Initial position and speed may be placed at random position
- Each particle may have a different life time

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

24/139

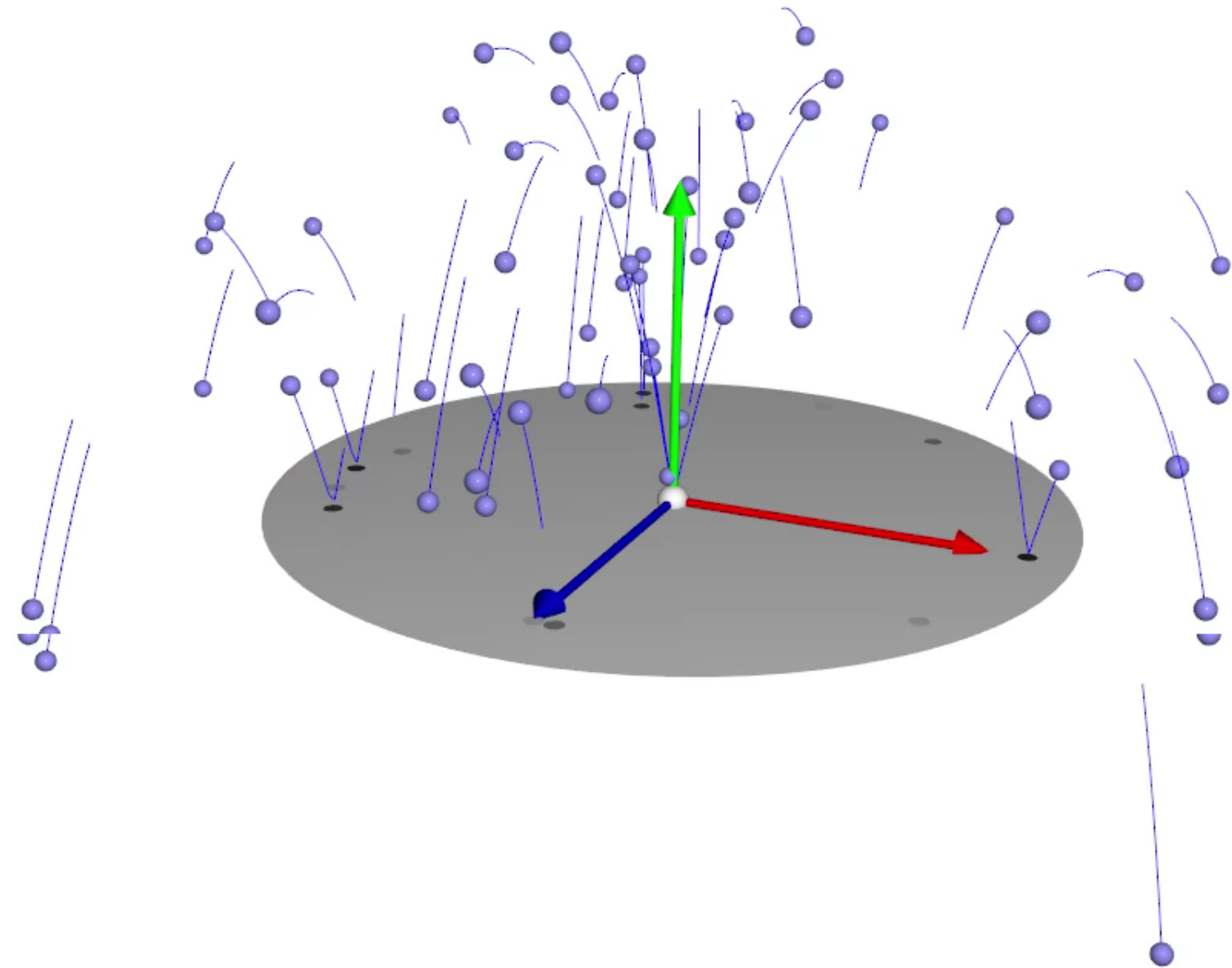
Q. What is the equation of motion ?
(taking into account the bouncing)

$$p(t) = \dots$$

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 ?



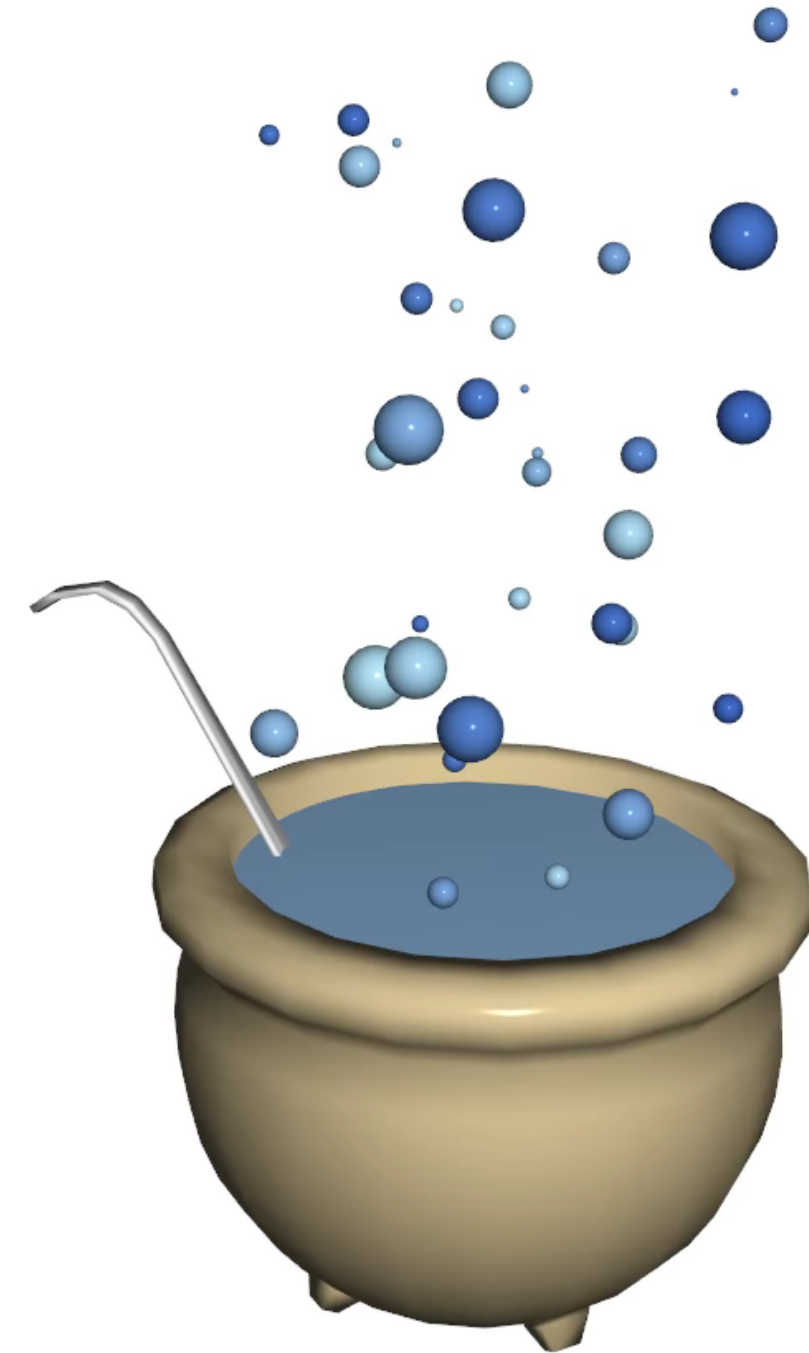
Manually defined/arbitrary motions

25/139

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

26/139

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)

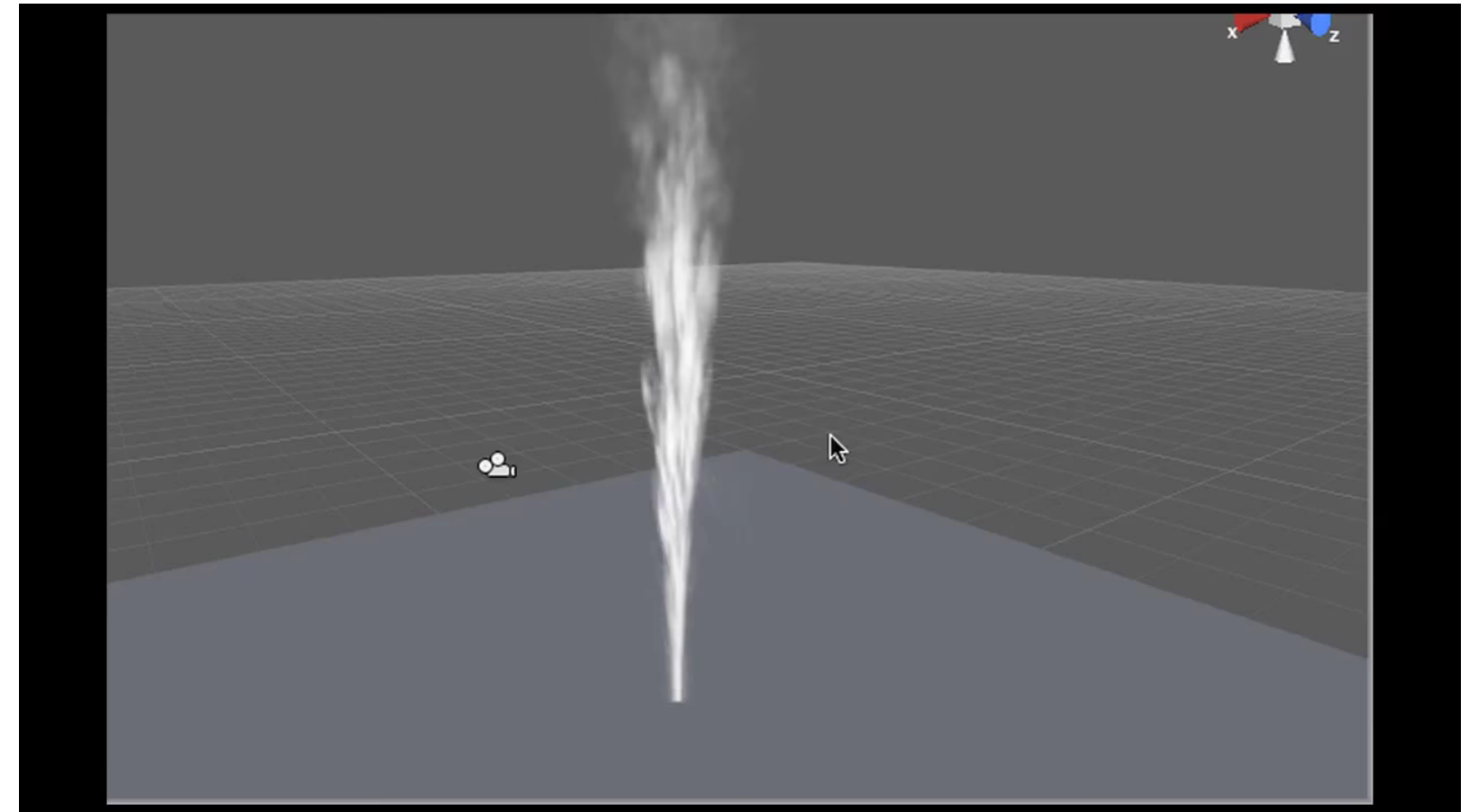
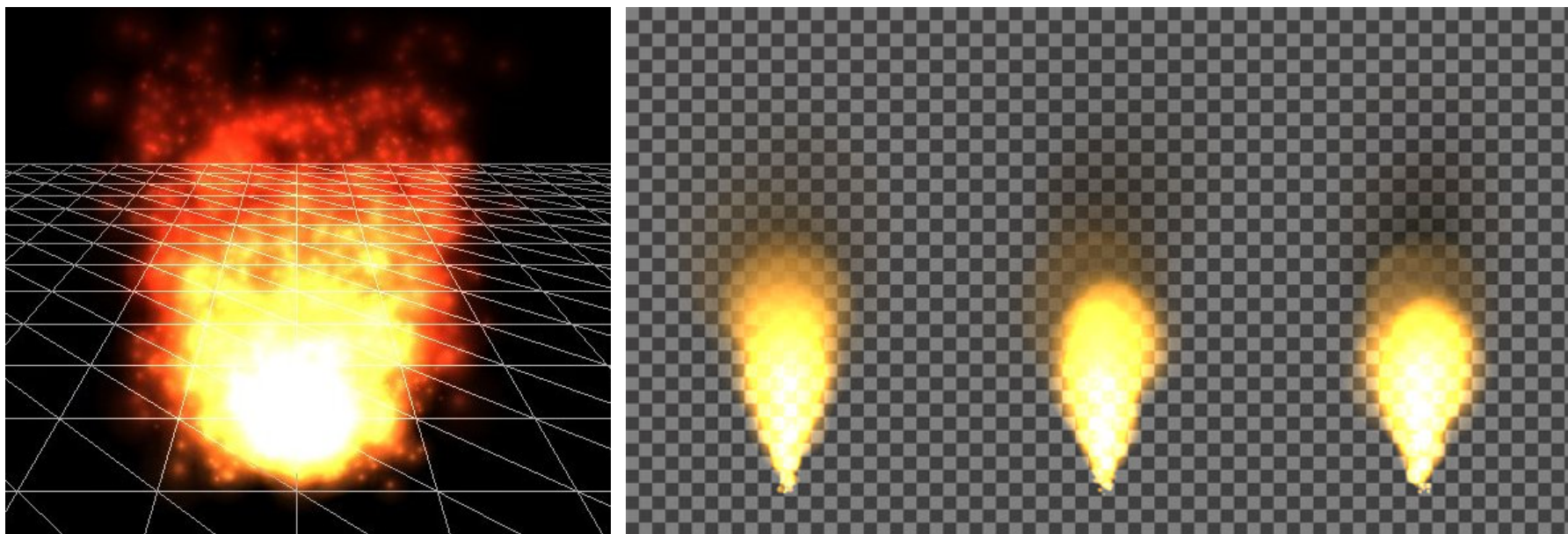
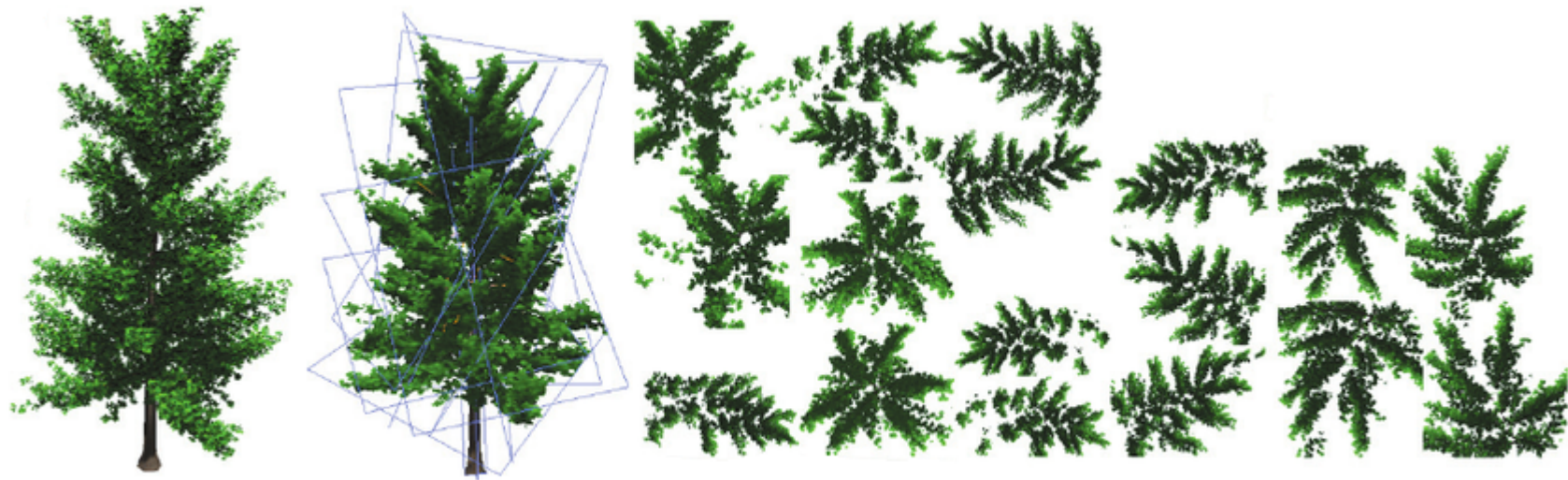


Usage of billboards

27/139

Large use of billboards for complex models

Vegetation, fire, smoke, etc.



Example of billboard

28/139

Follow up of the example

Q. How to model the following animation ?



Use case in production

29/139

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

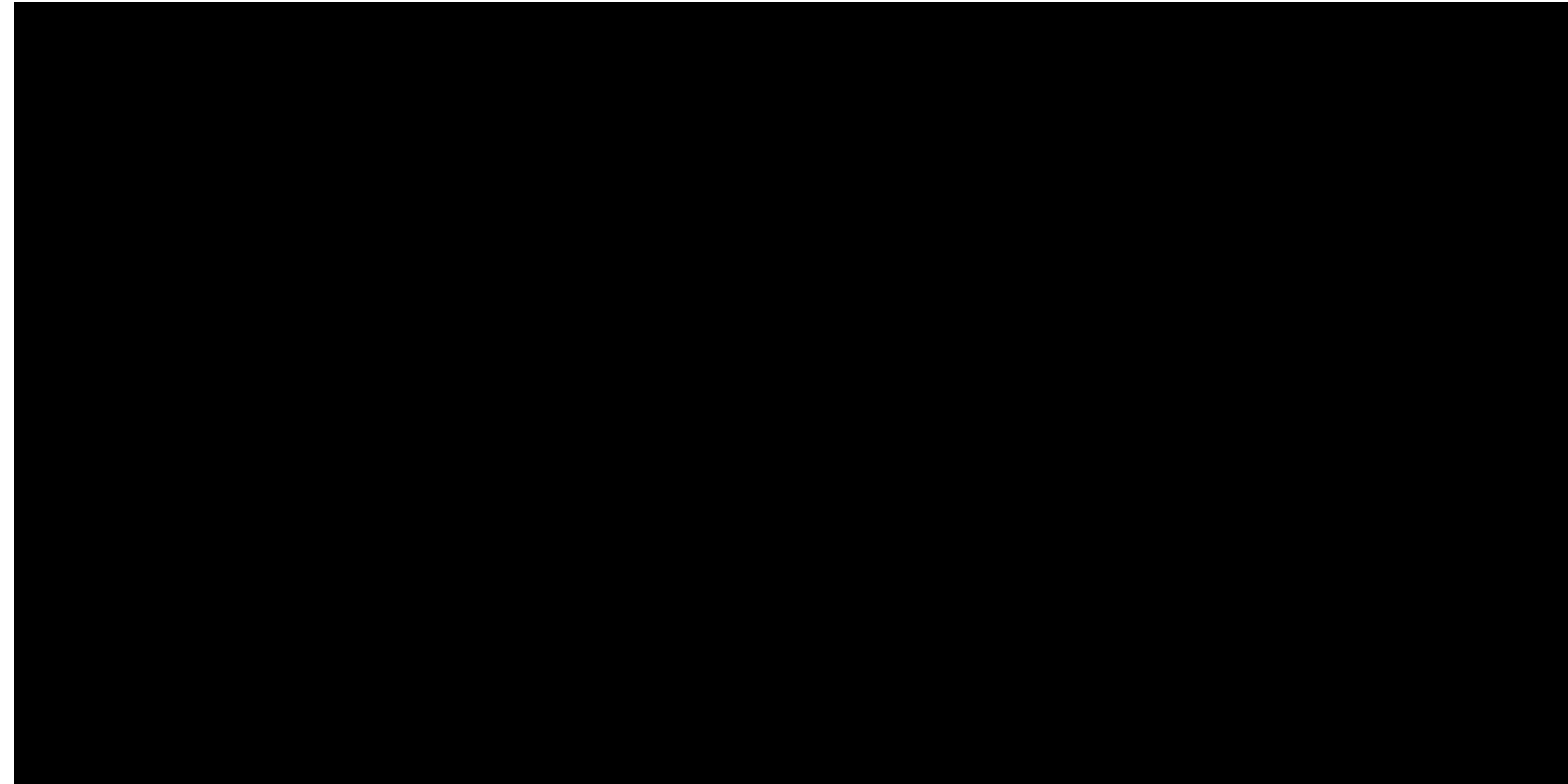


The Lord of the Rings

Use case in production

30/139

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

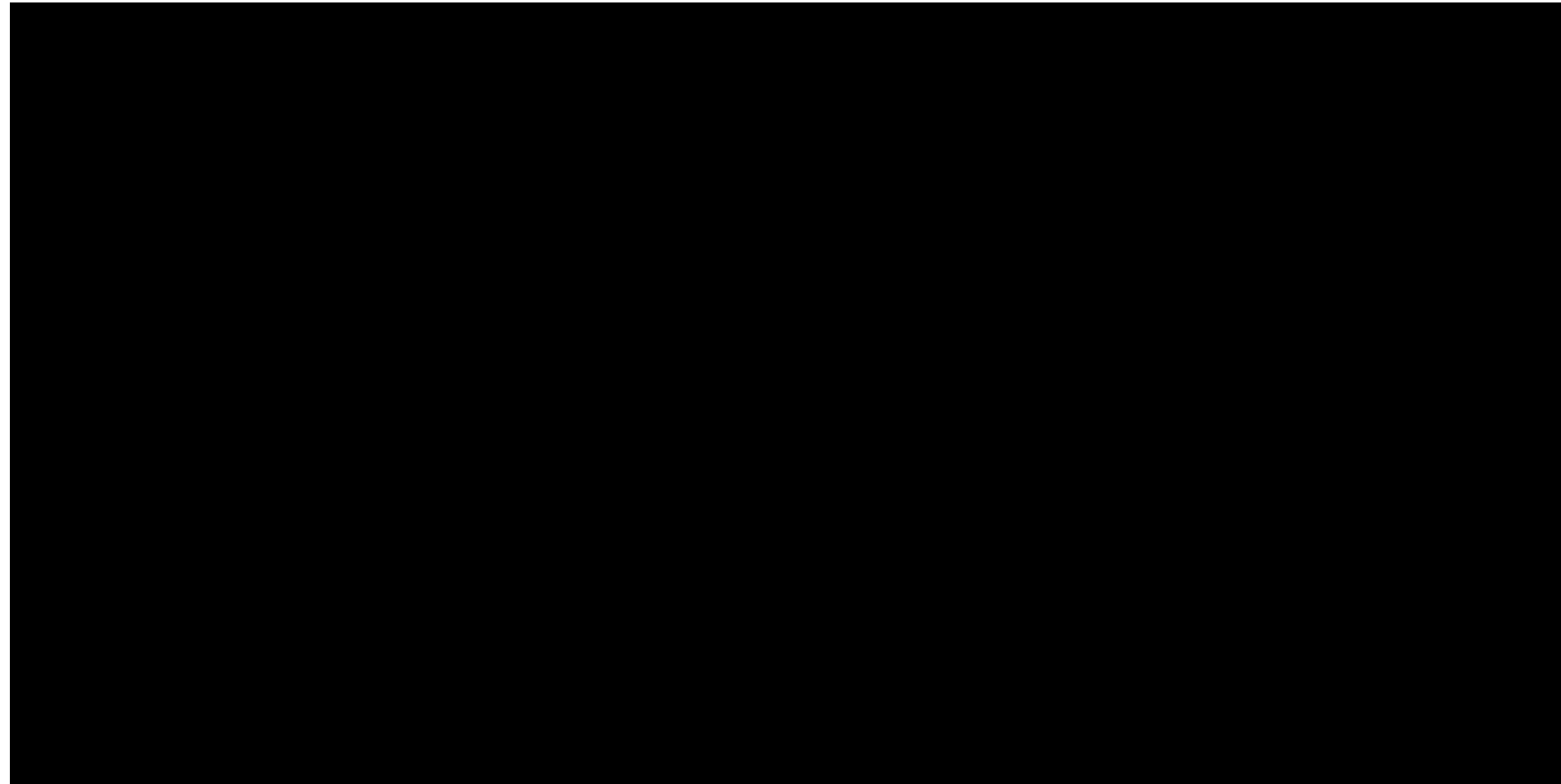


The Lord of the Rings

Use case in production

31/139

Full Making-Of



The Lord of the Rings

Implementation - transparency in OpenGL

32/139

Transparency is not straightforward using projection/rasterization rendering.

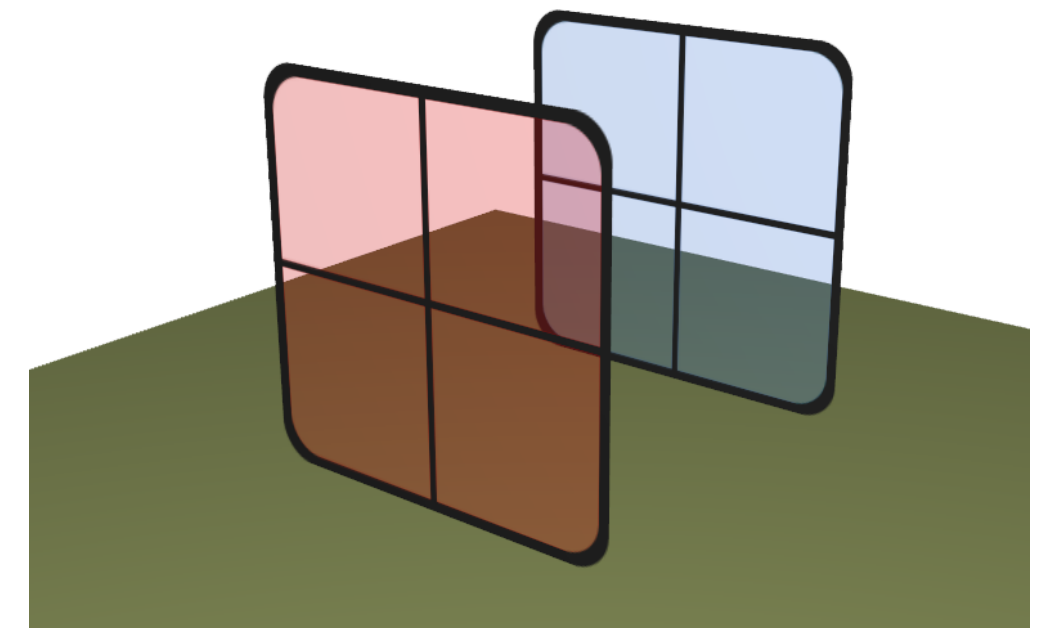
No perfect solution

⇒ 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

33/139

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);
```

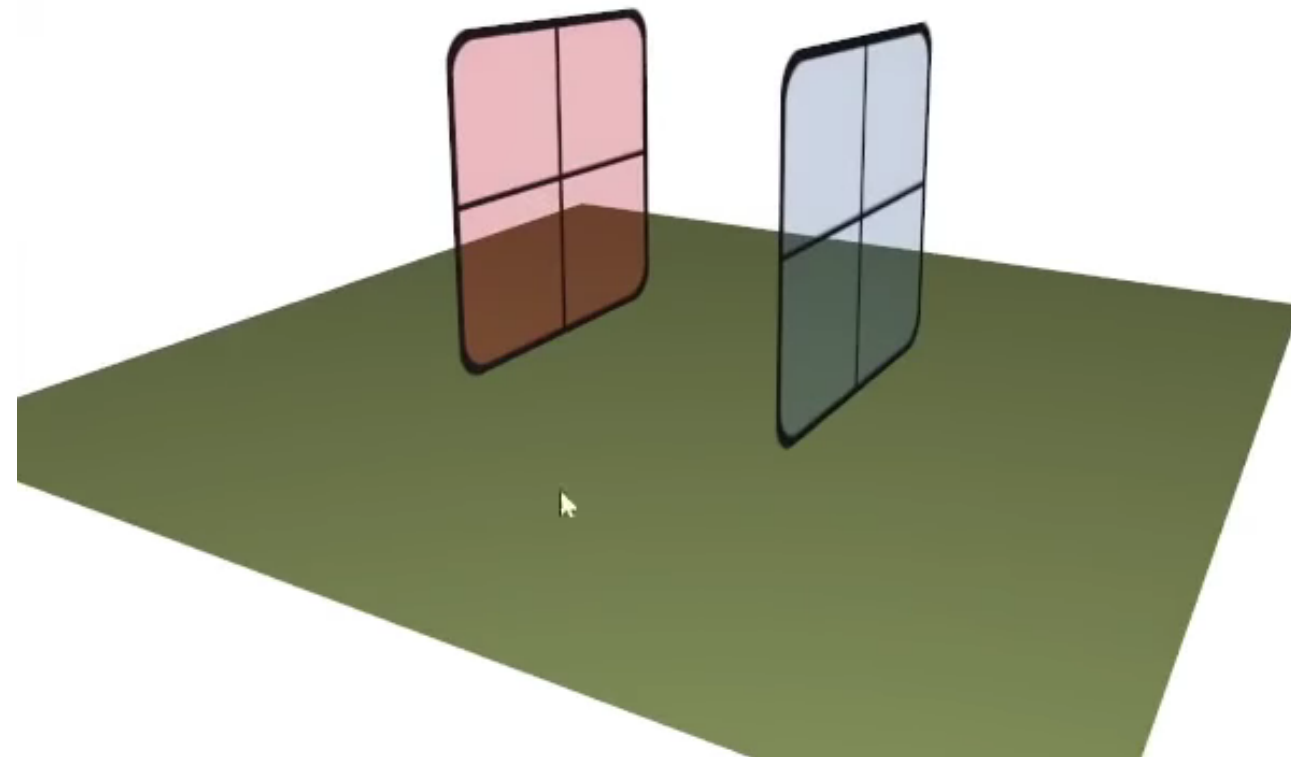
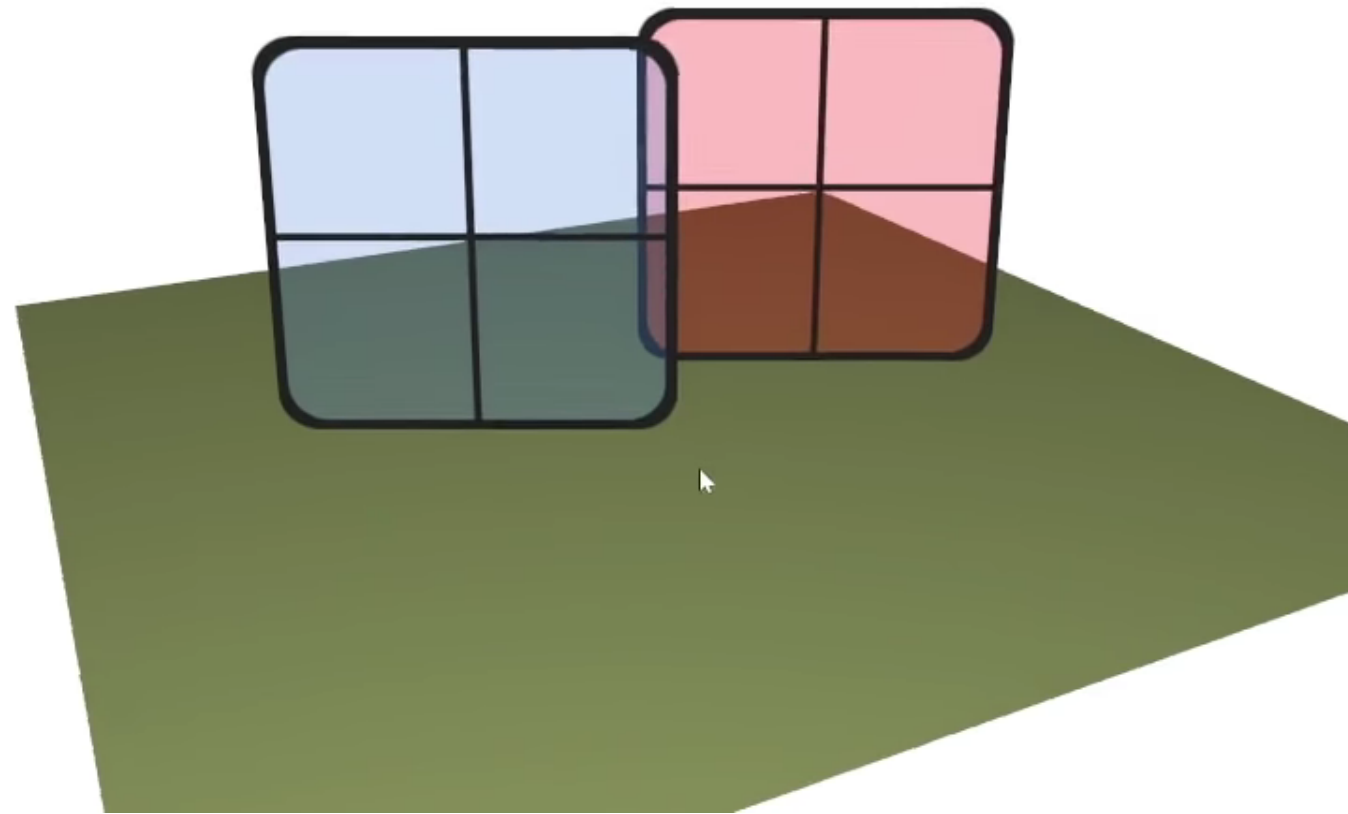


Image Blending - common errors

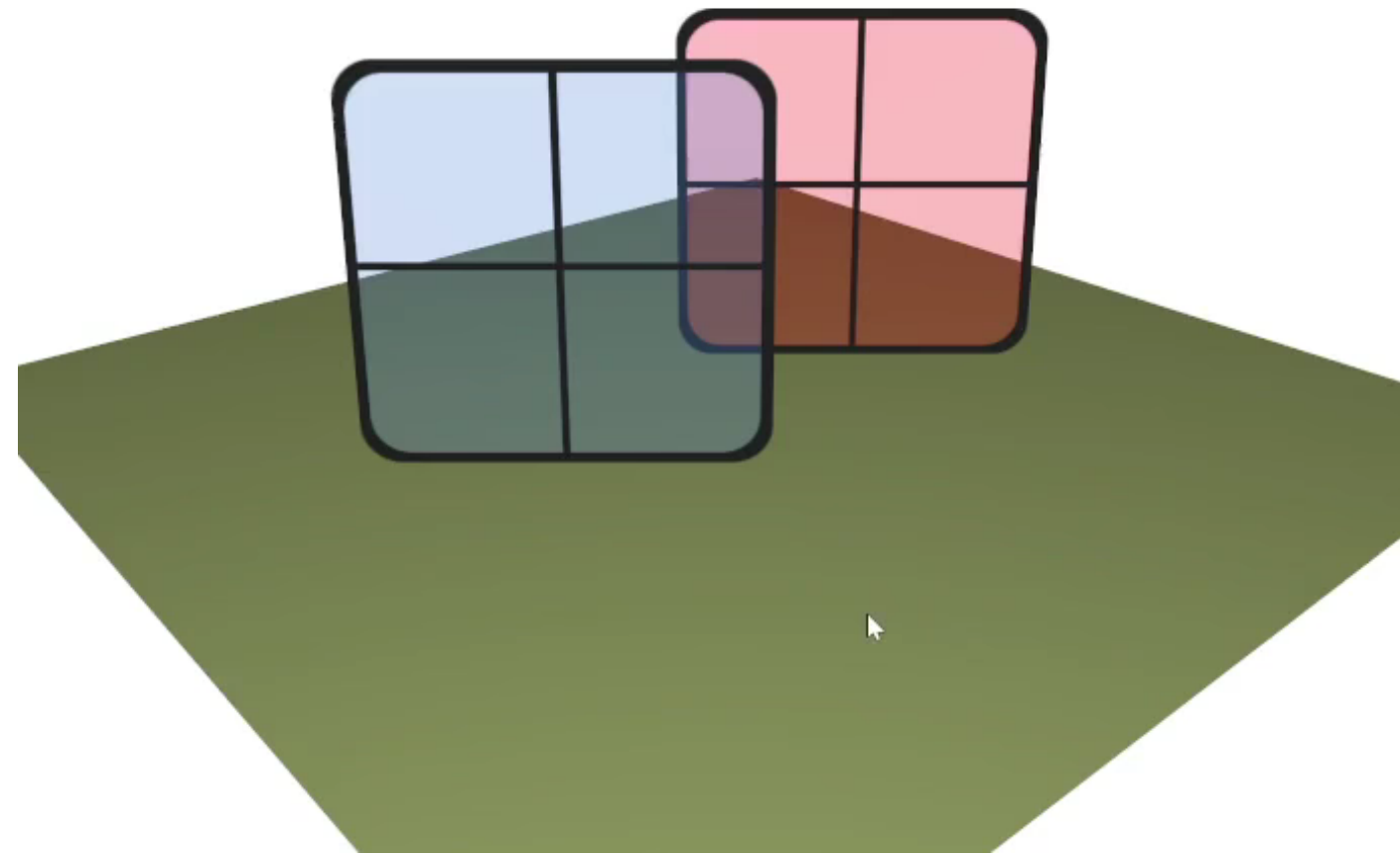
34/139

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

35/139

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

36/139

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](#)

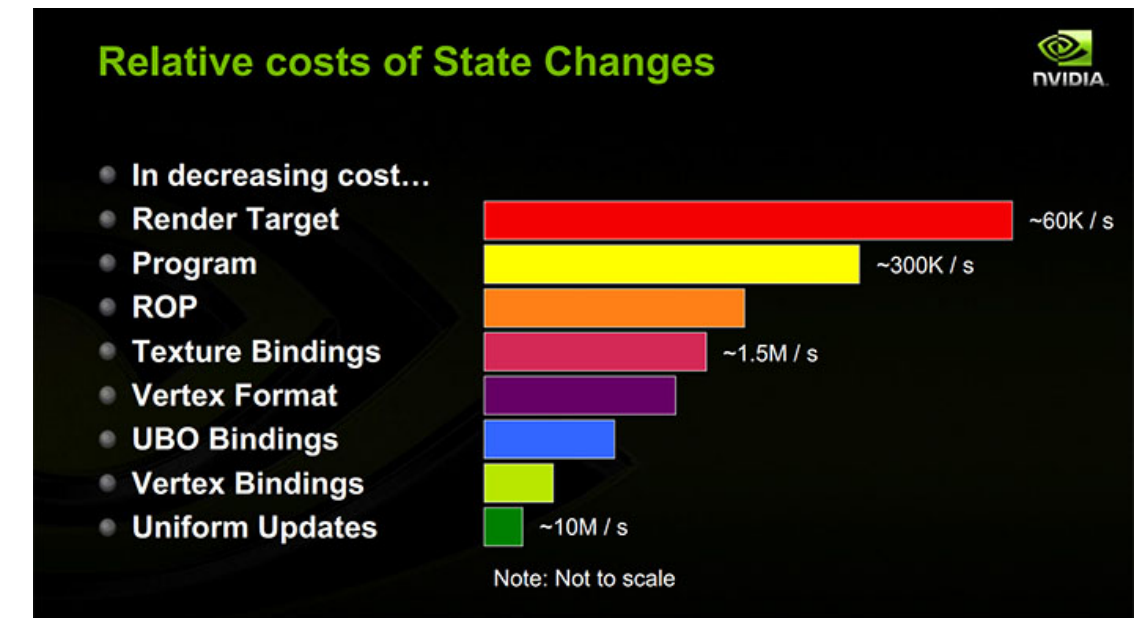
glDrawElements → [glDrawElementsInstanced](#)

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]

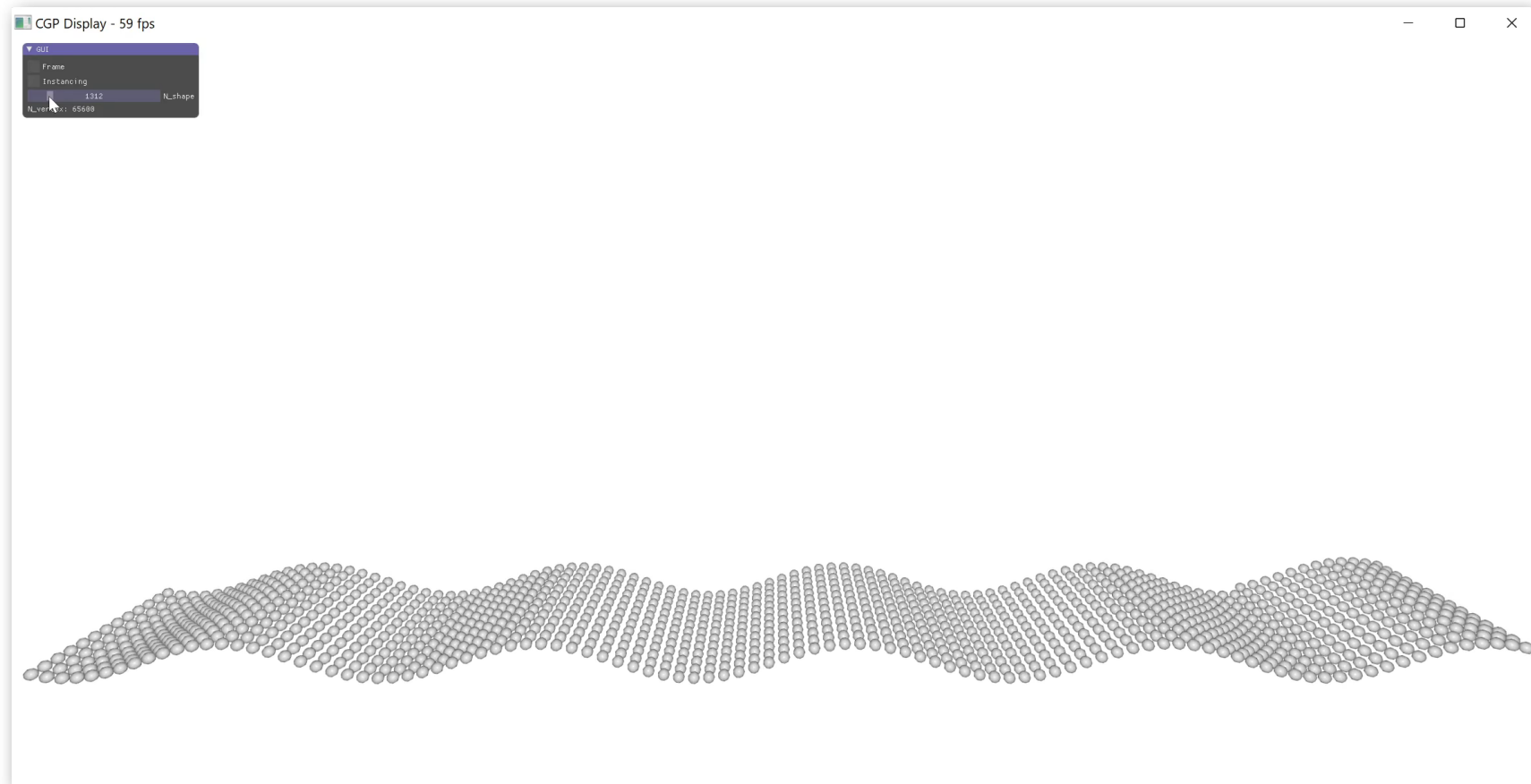
(+) Full capability of your GPU

(-) Require a specific shader

Instancing Demo

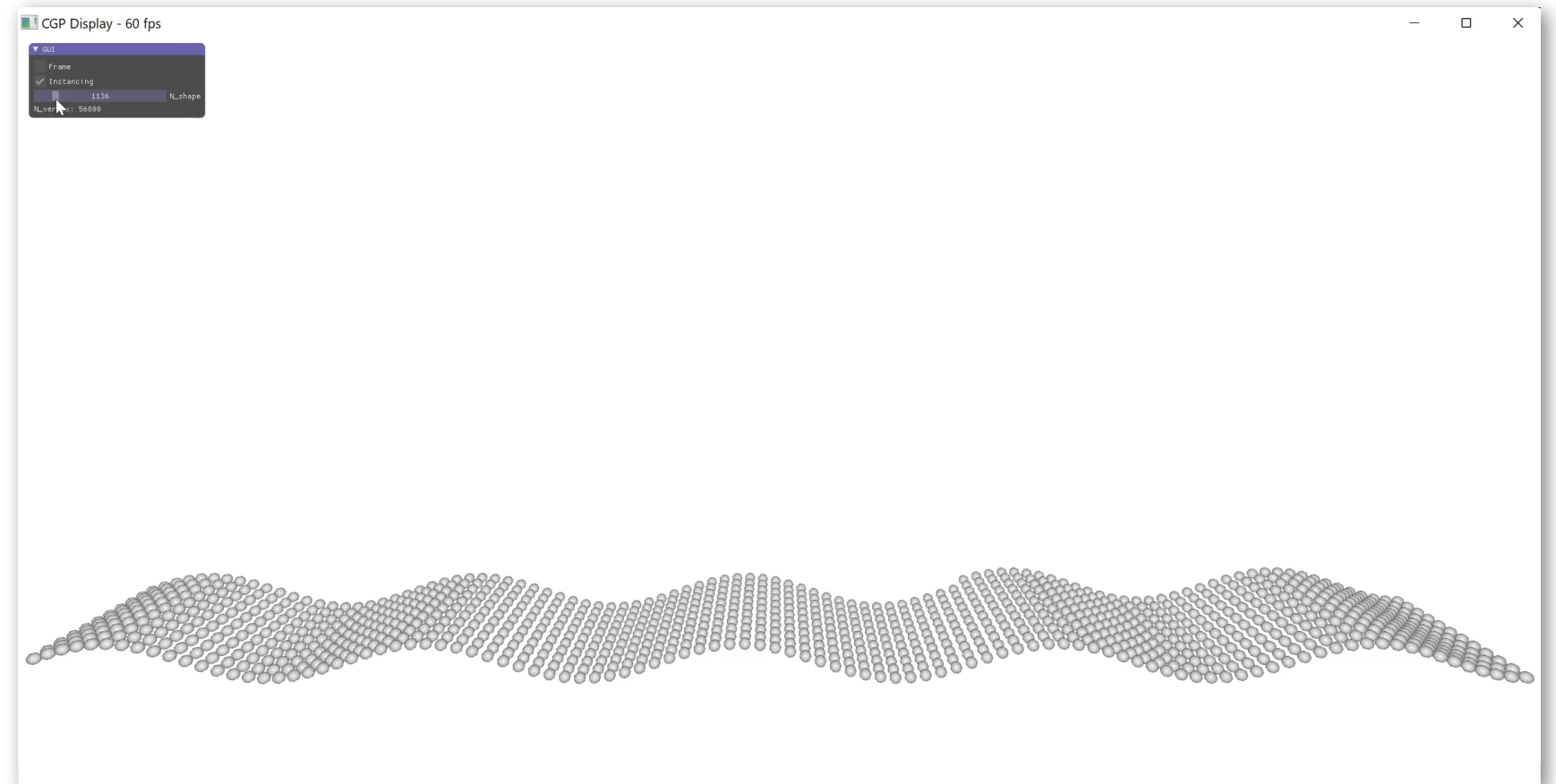
37/139

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

Affine Transformation

Affine/Linear Transformation

39/139

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

40/139

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

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

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

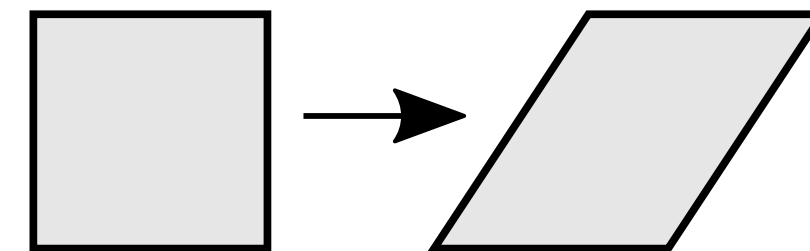
$$sh_{yx}(p) = (x, y + \lambda x, z)$$

...

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

Usually avoided in Graphics

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



Rotations

41/139

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

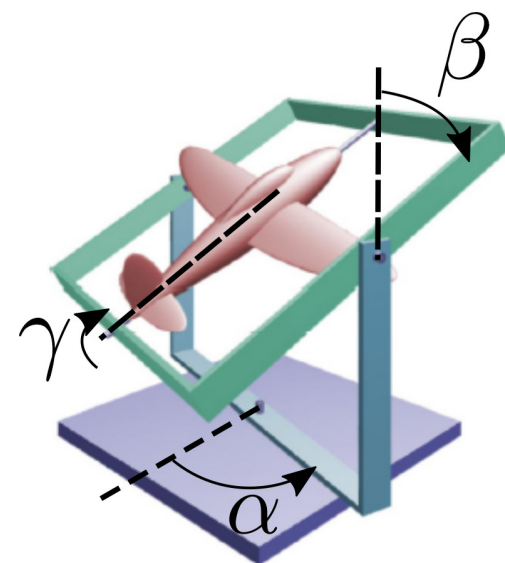
3 angles: (α, β, γ)

Composition of rotation around basic axes

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

(+) Meaningfull parameters

(-) Gimbal-lock

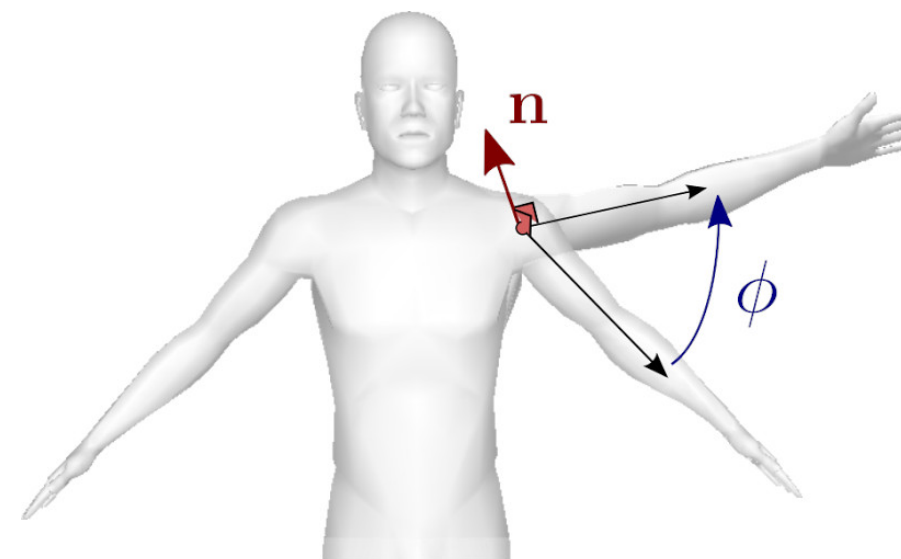


Axis angle

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

(+) 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

42/139

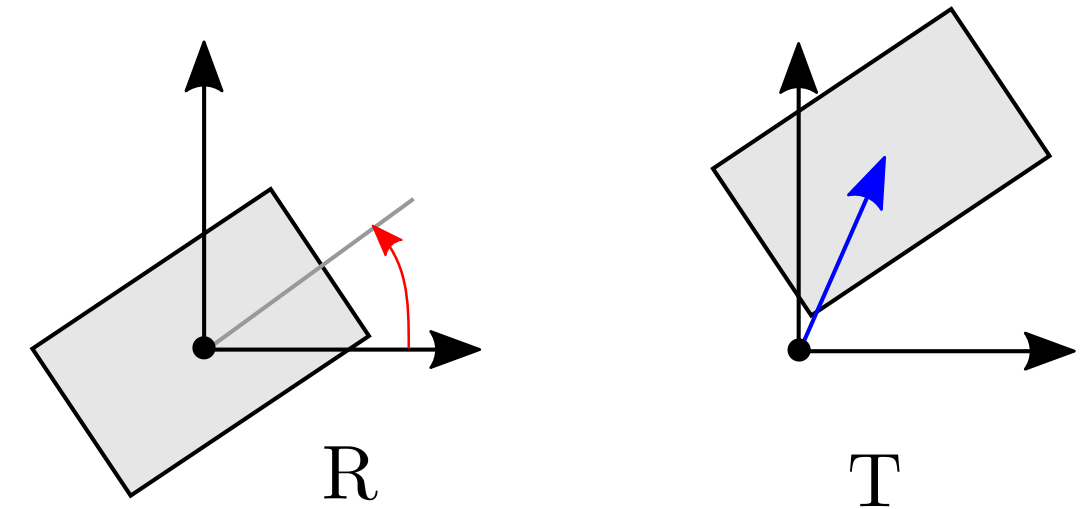
Take care, order of operation does matter !

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.

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

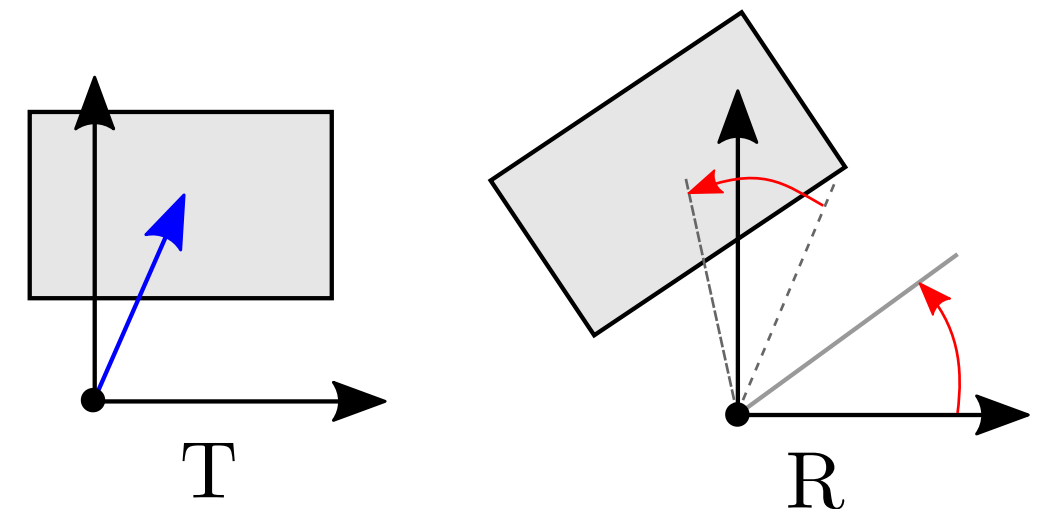
First rotates, then translates



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

First translates, then rotates

Rotation always happens around the origin.



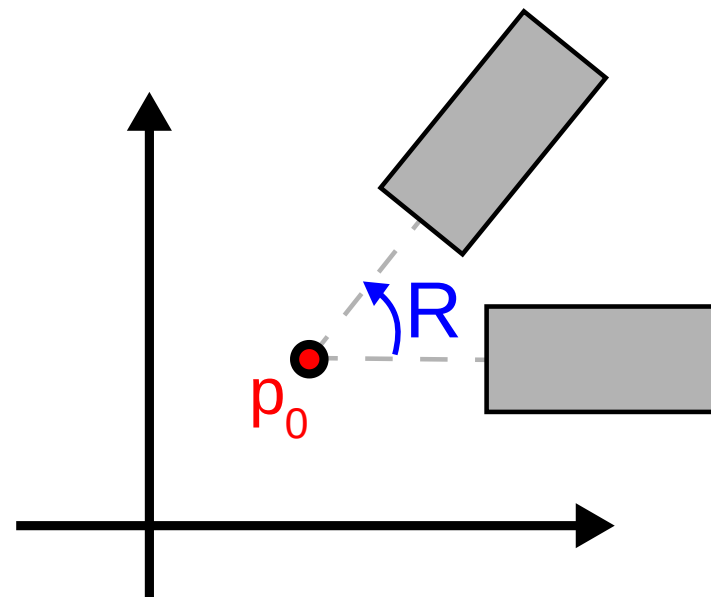
Beware: Some libraries (old OpenGL, Three.js) apply transformation from "left to right" using transposed matrix multiplication

Affine Transformation Exercises

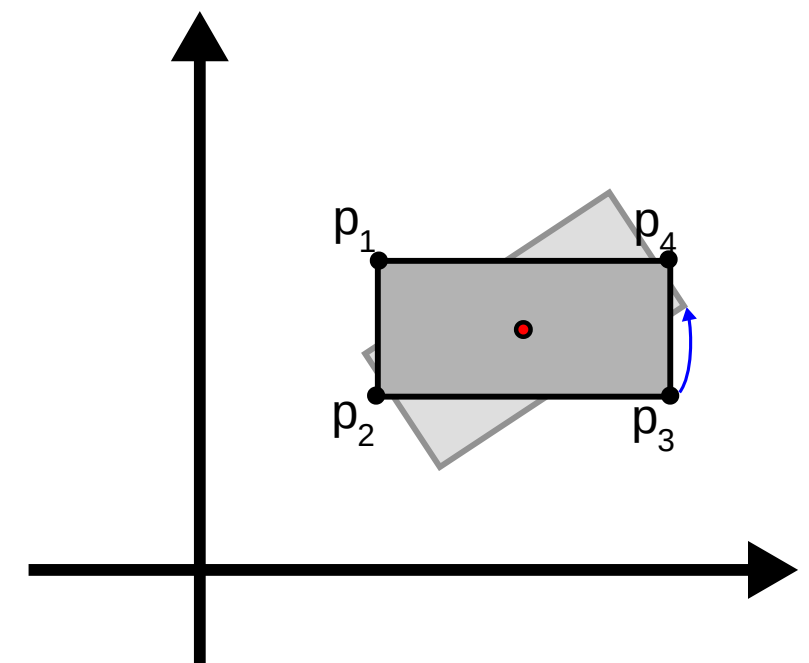
43/139

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.

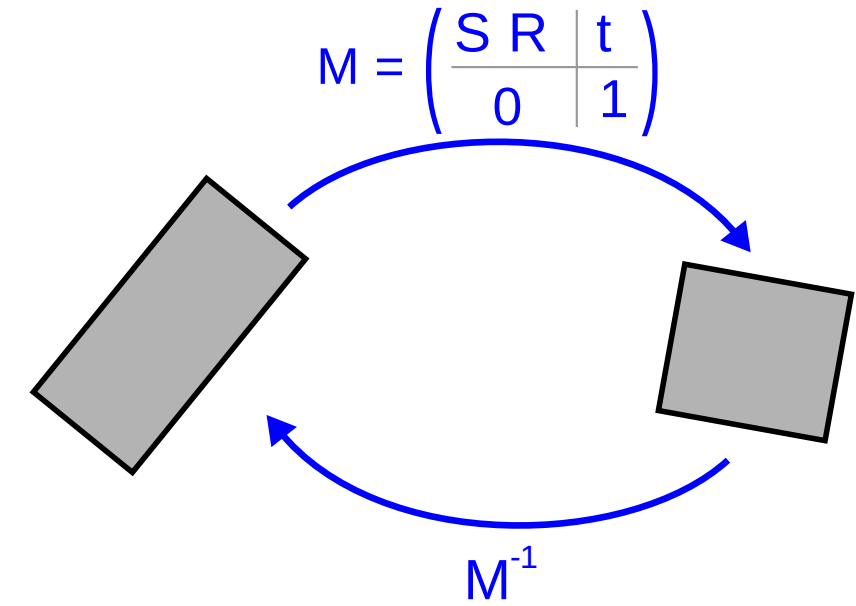


Affine Transformation Exercises

44/139

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

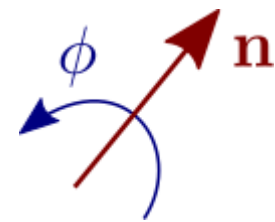


Rotation: Focus Axis Angle

45/139

Any 3D rotation can be represented by

- A unit axis n
- An angle θ

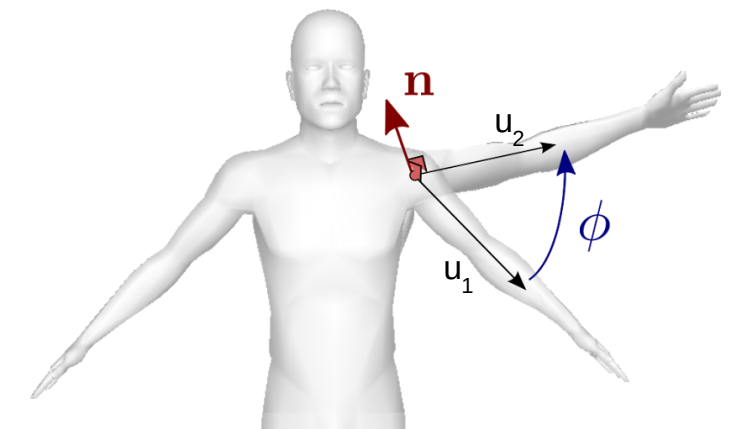
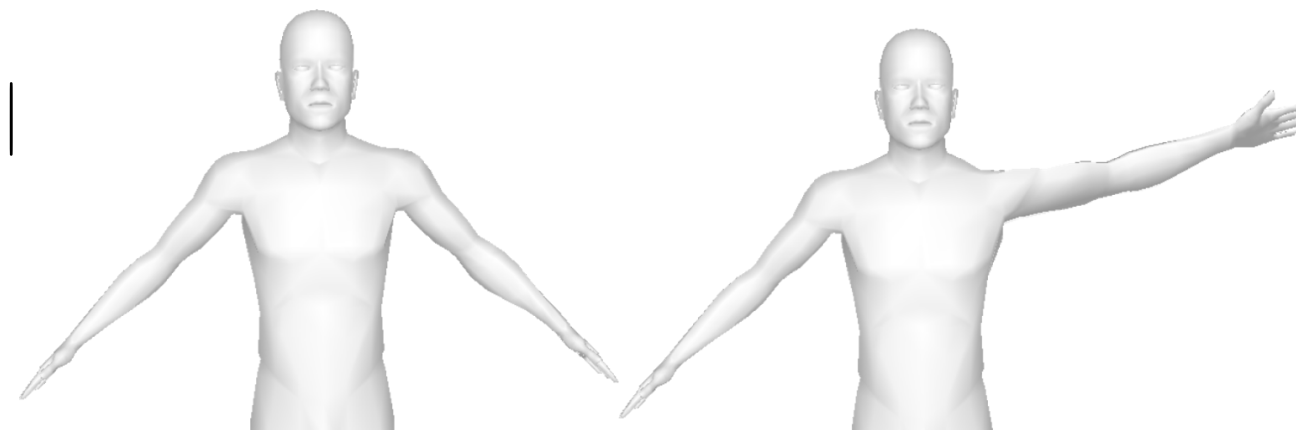


Pro.

- 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

46/139

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

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

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

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

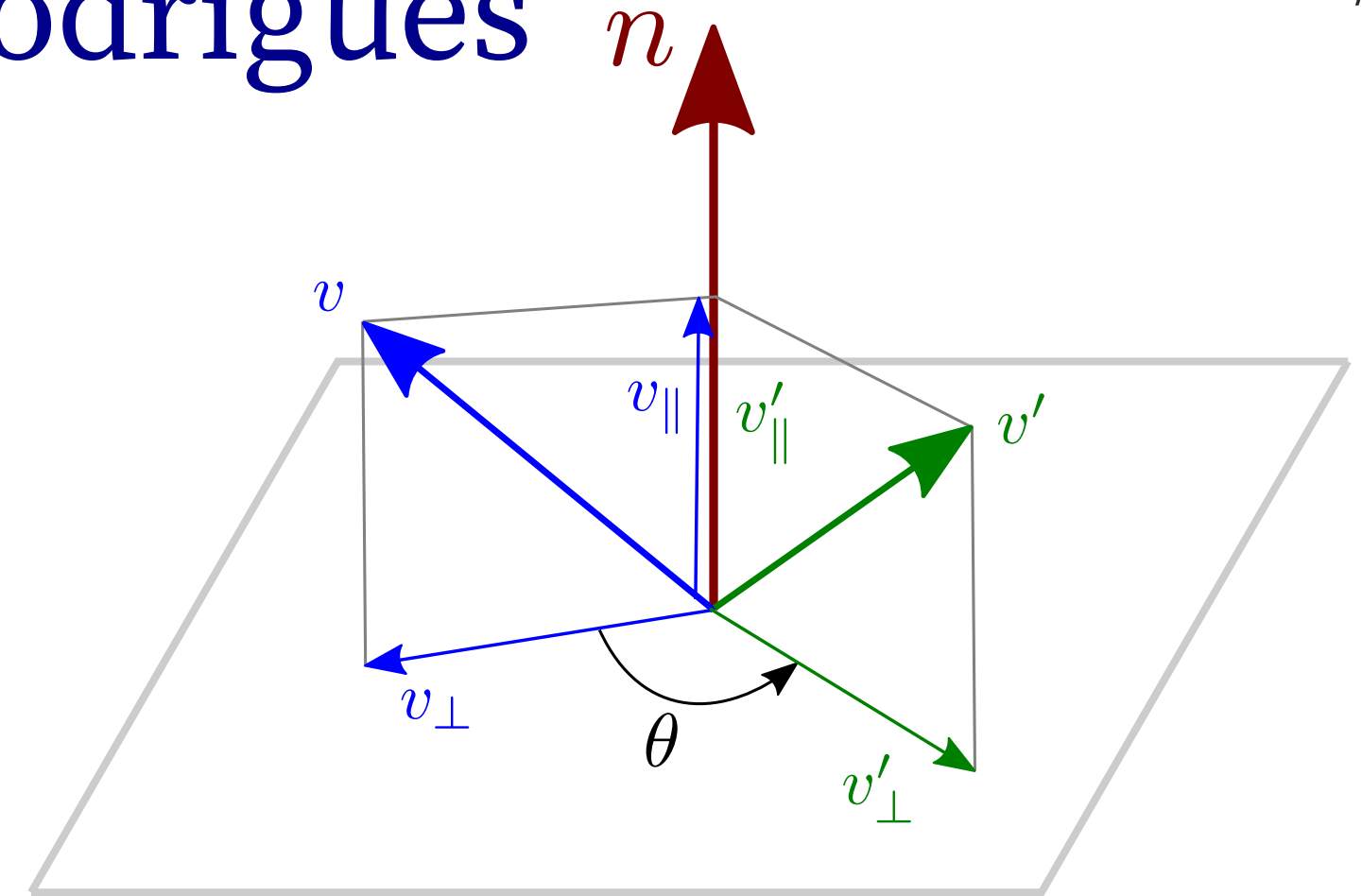
$$v_{\parallel} = (v \cdot n) n$$

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

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

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

Rodrigues' rotation Formula



Rotation: Focus Axis Angle as Matrix

47/139

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

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

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

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

Rotation: Focus Axis Angle - Summary

48/139

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

49/139

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

- 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

51/139

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

52/139

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

53/139

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

54/139

Representation and Operations

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

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

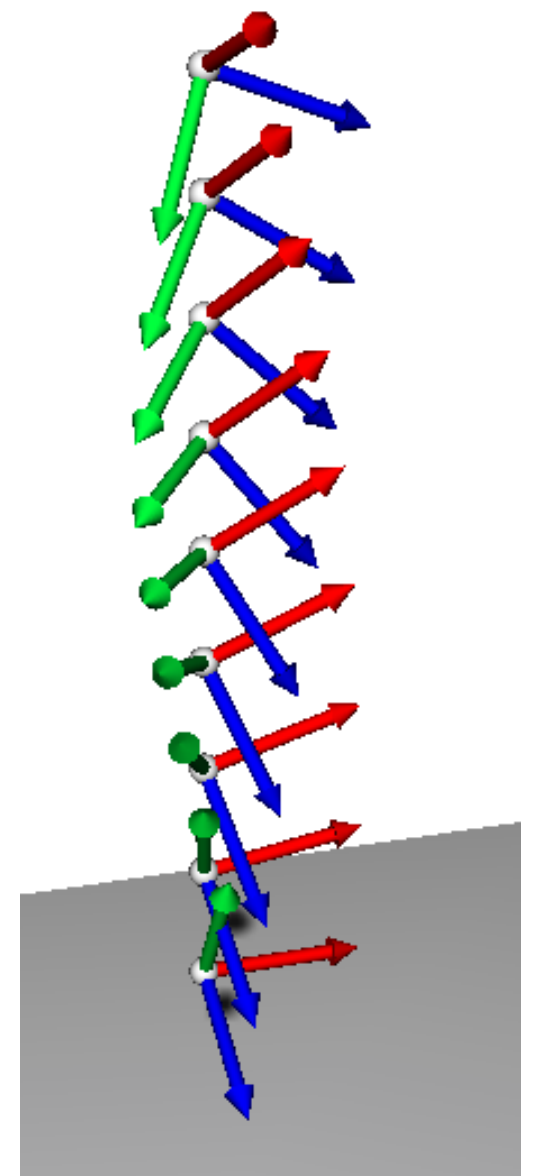
$$q(t) = \frac{\sum_j \alpha_j q_j}{\|\sum_j \alpha_j q_j\|}$$

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



Shape Deformation (I)

Volume-based deformation

Parametric deformation

Parametric deformation

57/139

Use succession of parameterized space varying deformation

Rotation: Bending, Twisting

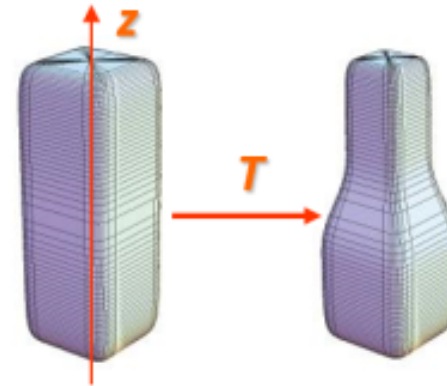
Scaling: Squeezing/tapering, Stretch

Old idea: First proposed in

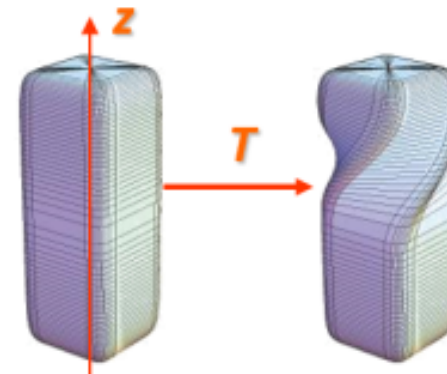
[Alan H. Barr. *Global and Local Deformations of Solid Primitives*. SIGGRAPH 1984]

(+) Explicit formulation $q = f(p)$

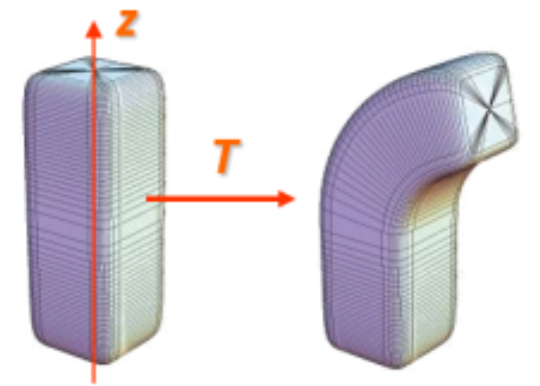
(-) Limited degrees of freedom



$$\begin{pmatrix} s(z) & 0 & 0 & 0 \\ 0 & s(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$



$$\begin{pmatrix} \cos \theta(z) & \sin \theta(z) & 0 & 0 \\ -\sin \theta(z) & \cos \theta(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$



$$\begin{pmatrix} \cos \theta(z) & 0 & -\sin \theta(z) & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta(z) & 0 & \cos \theta(z) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

Local deformers

58/139

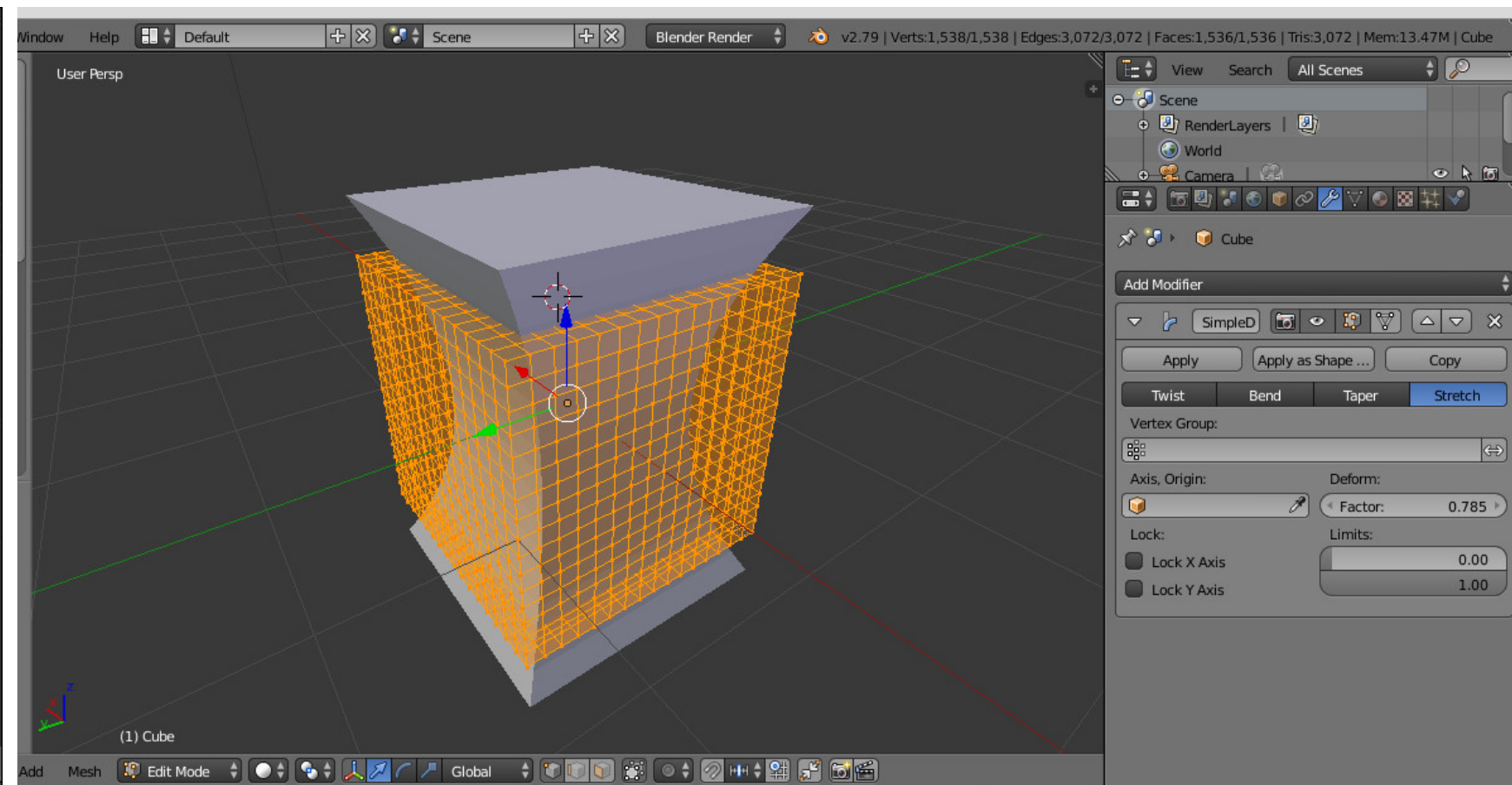
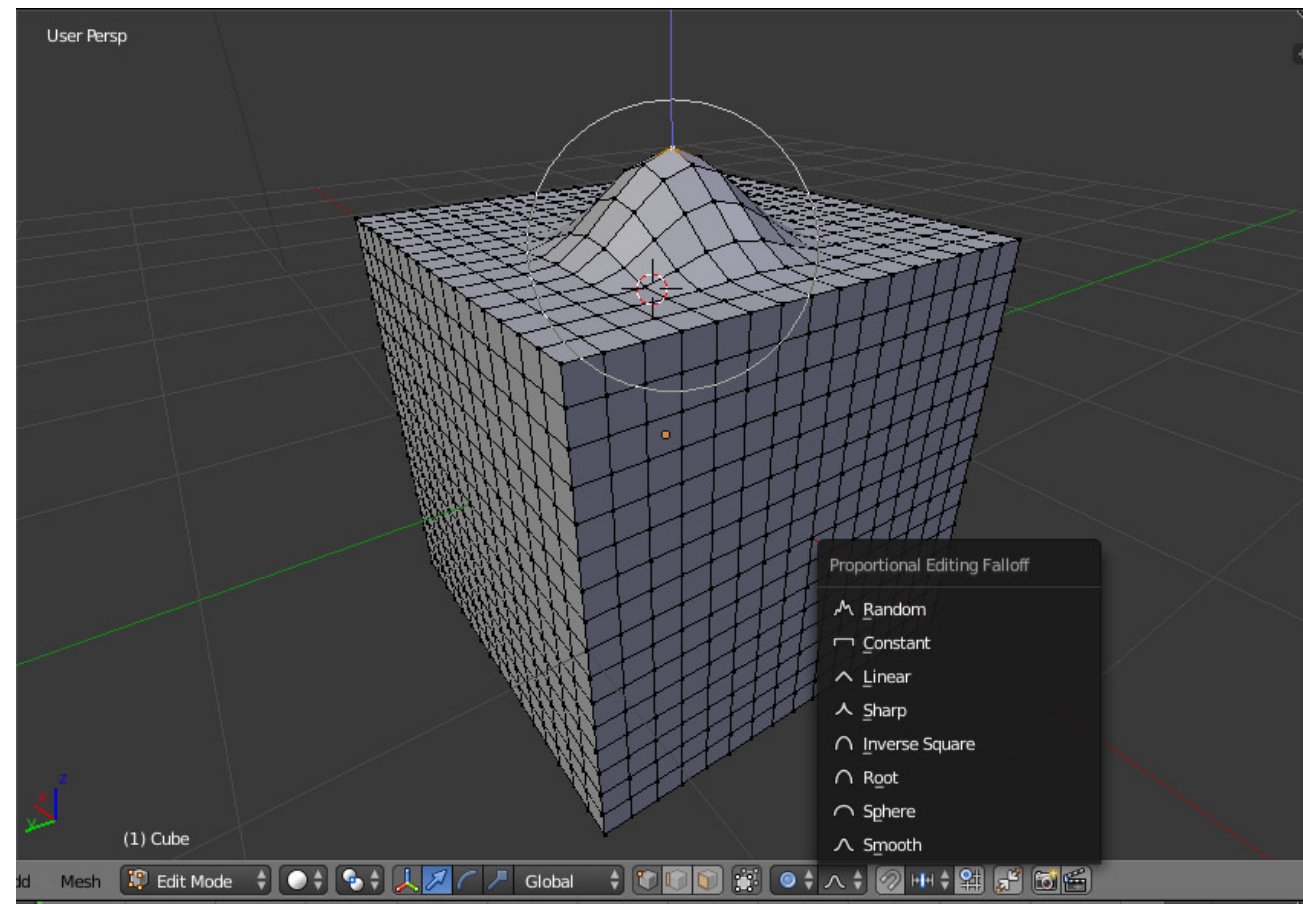
Variety of interactive local and parametric **deformers** in 3D modeling softwares

Example: Applying a translation t only locally around c

$$q = p + w(p) t, \quad w(p) = \exp(-\|p - c\|^2 / \sigma^2)$$

t can be controlled interactively

and other parameters (rotation axis, angle, etc) as well.



Deformers example cases

59/139

Q. How to model this interactive deformer ?



Input:

- c : Selected position on the surface
- t_{2D} : 2D displacement coordinates of the mouse on screen while dragging
- r : Circle radius

Computation:

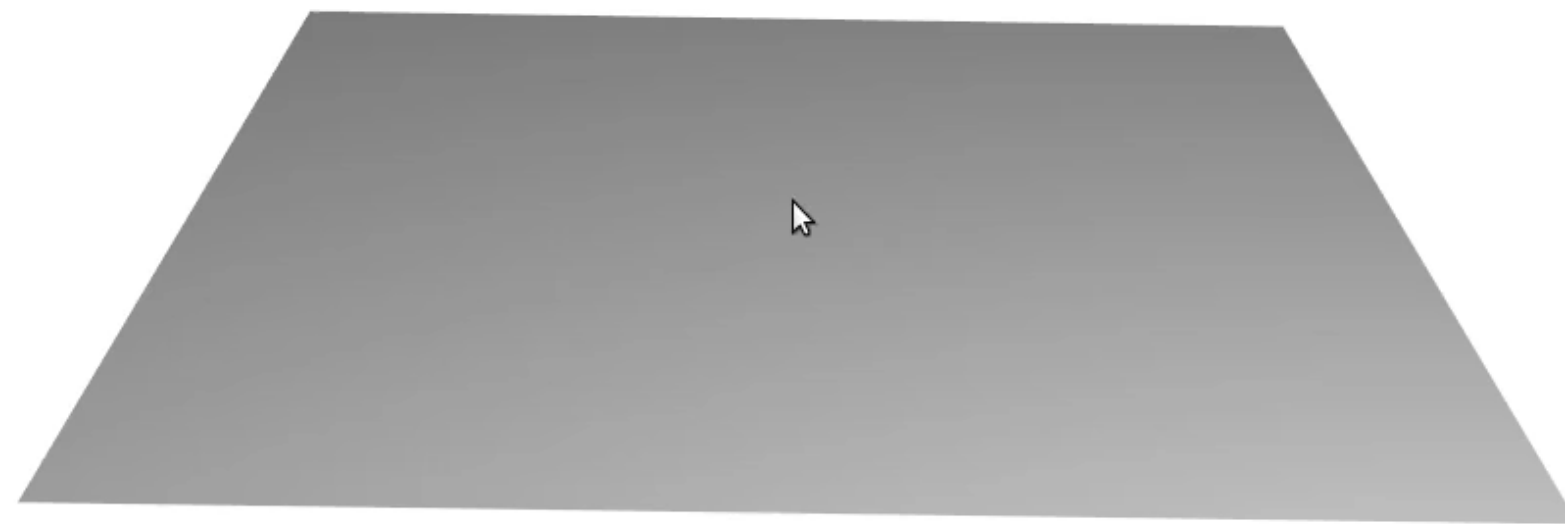
For all positions of the surface p

...

Deformers example cases

60/139

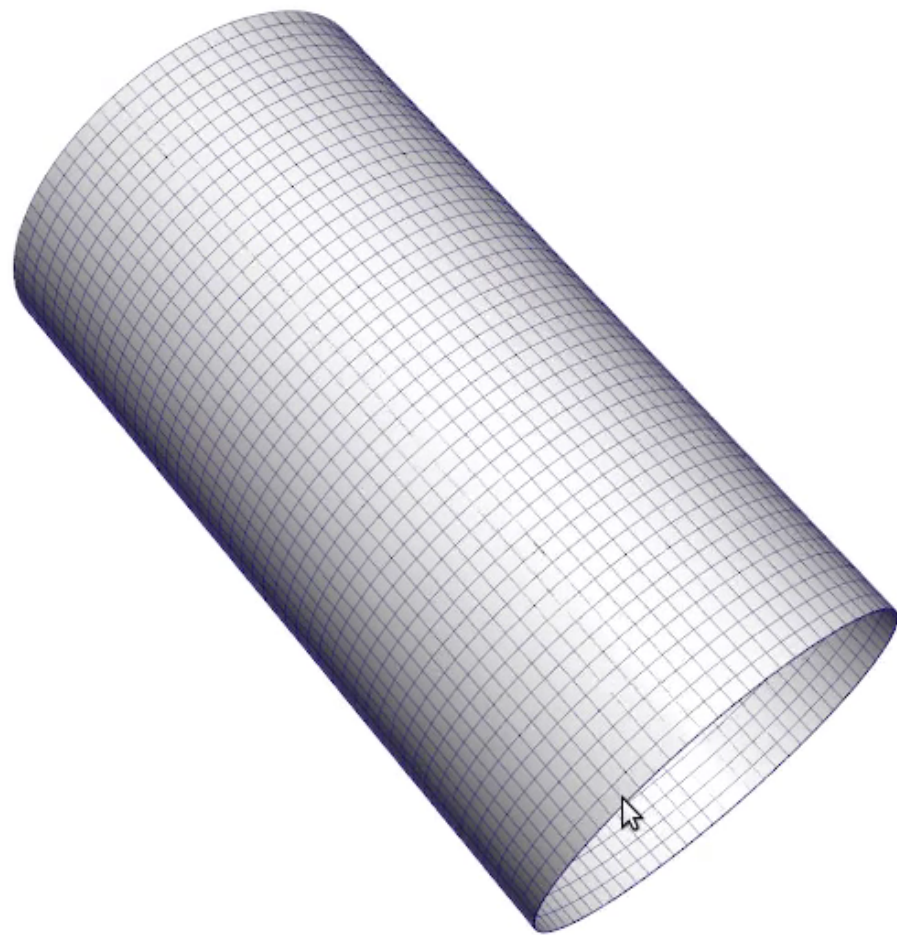
Q. How to model this interactive deformer ?



Deformers example cases

61/139

Q. How to model this interactive deformer ?



0:00 / 0:57

Procedural Animation

Noise function - Perlin Noise

Procedural Noise

63/139

What is a spatial/temporal procedural noise:

- Function with visible structure/pattern (limited frequency bandwidth)
- Without visible periodicity
- Deterministic (Same result from a given input)



Examples of 2D procedural textures

Create procedural noise

64/139

Ex. Continuous function $f(x)$ with limited frequency.

For integer value: $f(n) = \text{pseudo-random, deterministic, value}$

ex. Hash Function: `float hash(float n) { return fract(sin(n) * 1e4); }`

Interpolate using smooth curve between each integer value (Smooth step, cubic polynomials, etc.)

Properties:

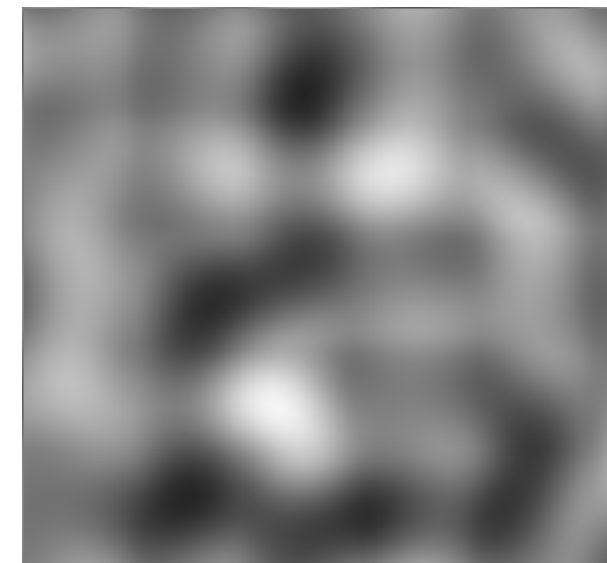
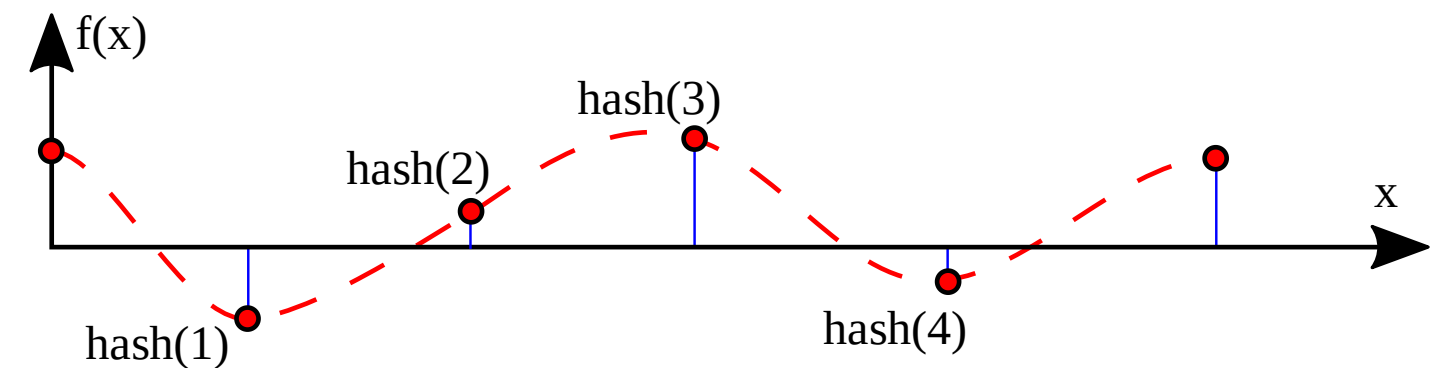
- Continuous function
- Looks non-periodic
- Frequency limited around 1

Can be computed in 2D, 3D, etc.

Simple algorithm - ex. in 1D

For a given x , $n = \text{floor}(x)$

- Evaluate hash function at n , $n + 1$
($(n \pm i)$ for further sampling)
- Compute interpolated value $f(x)$ at position x



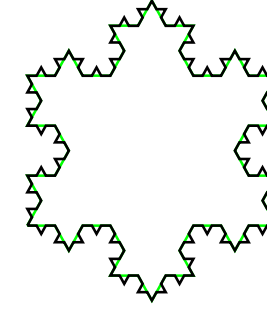
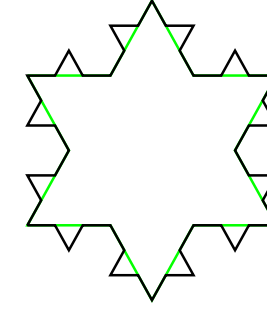
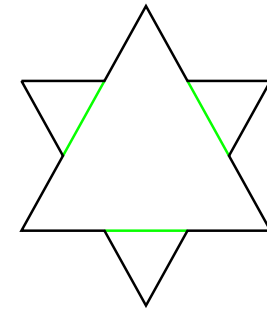
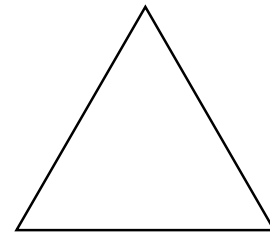
Add high frequency details: Notion of Fractal

65/139

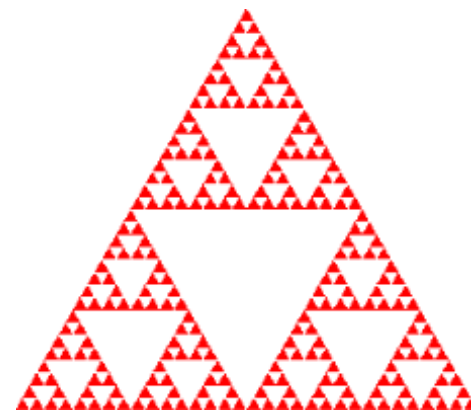
Idea: Recursively add self-similar details

Simple rule $\Rightarrow$ complex shapes

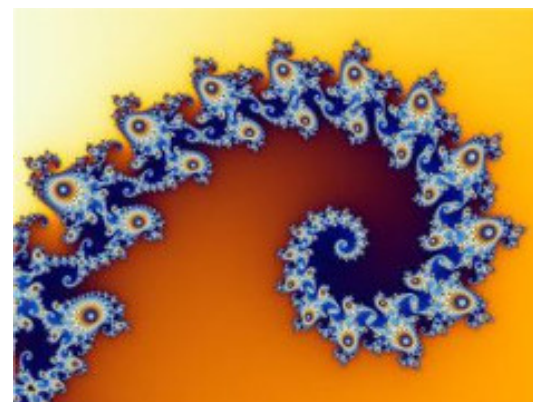
May look like complex natural details



Koch Snowflake



Sierpinski triangle, Shell:Oliva Porphyria



Standard fractals (Mandelbrots, Sierpinski, Newtons's root, etc) are very hard to control.

Not often used directly in CG.

Perlin Noise

First procedural noise proposed in

[Ken Perlin, *An Image Synthesizer*. SIGGRAPH 85]

Idea: Sum over pseudo-random function with increasing frequencies and decreasing magnitude

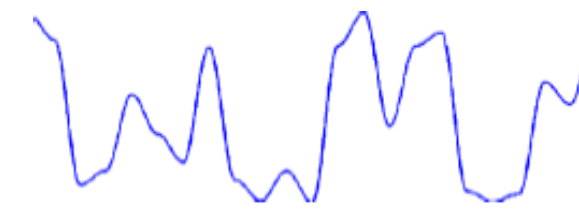
f : Smooth pseudo-random function

$$P(x) = \sum_{k=0}^N \alpha^k f(\omega^k x)$$

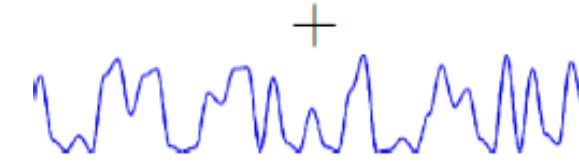
- N number of Octave

- α persistency ($1/\alpha$ attenuation)

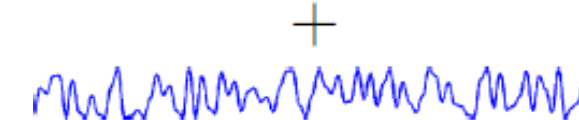
- ω frequency gain



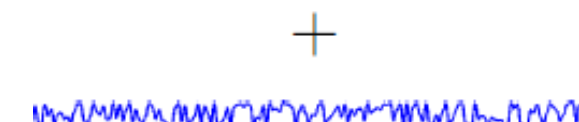
$f(x)$



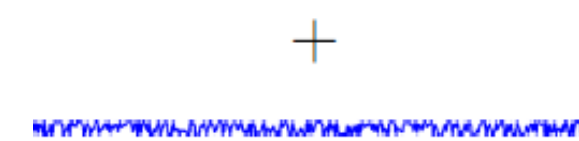
$0.8 f(2x)$



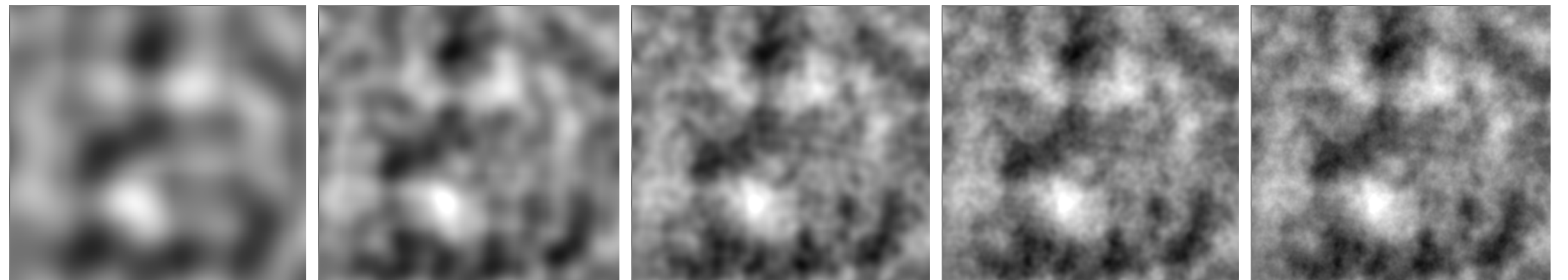
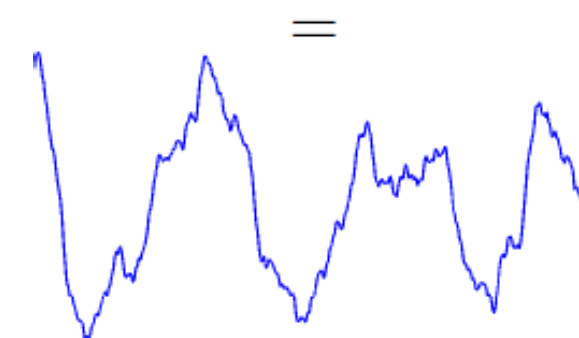
$0.8^2 f(4x)$



$0.8^3 f(8x)$



$0.8^4 f(16x)$

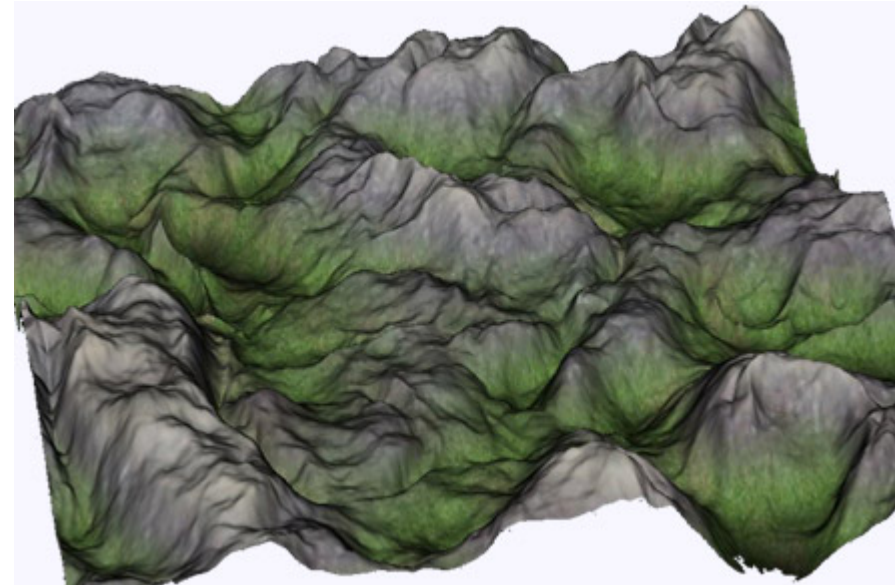


Perlin noise usage

67/139

Direct use: $z = P(x, y)$

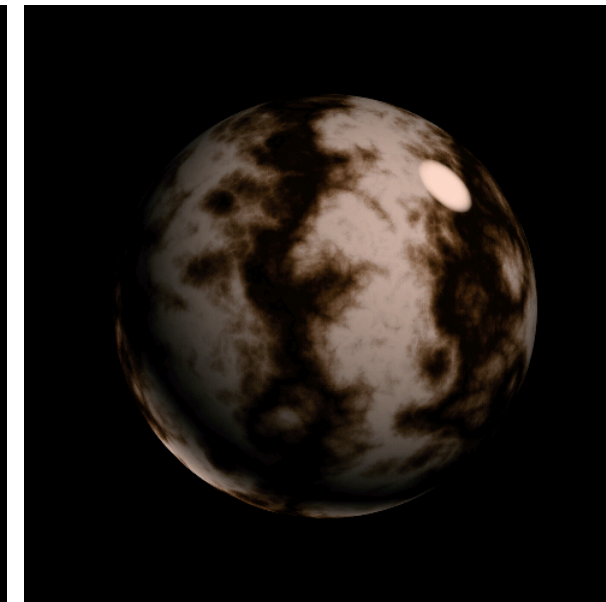
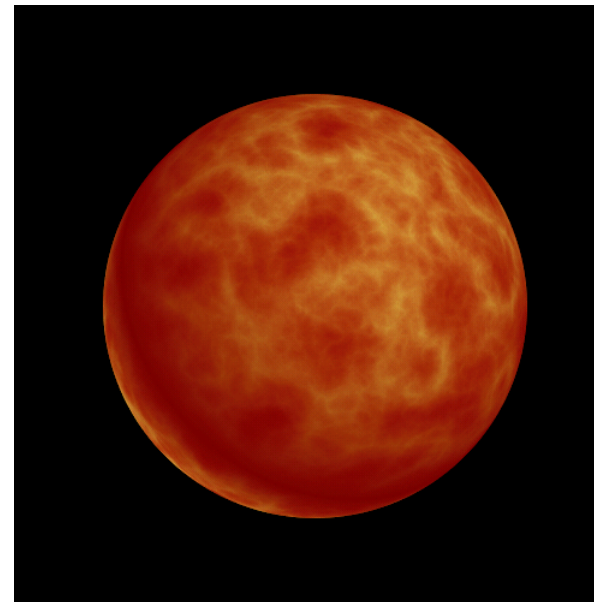
Mountain-looking terrain



Material texture

Ridge effect: $\sum_k \alpha^k |f(\omega^k x)|$

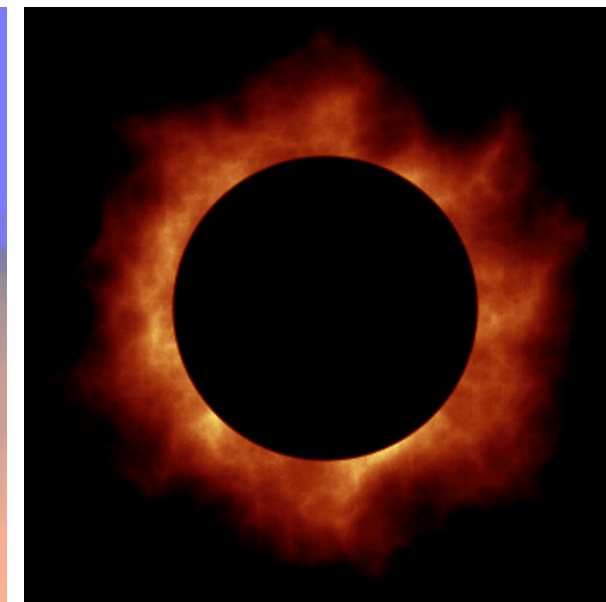
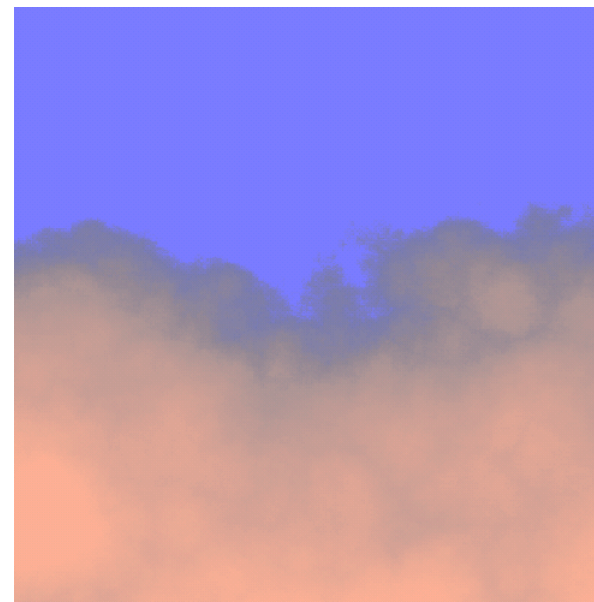
Marble effect: $\sin(x + \sum_k \alpha^k |f(\omega^k x)|)$



Animated textures

Translation: $P(x, y + t)$

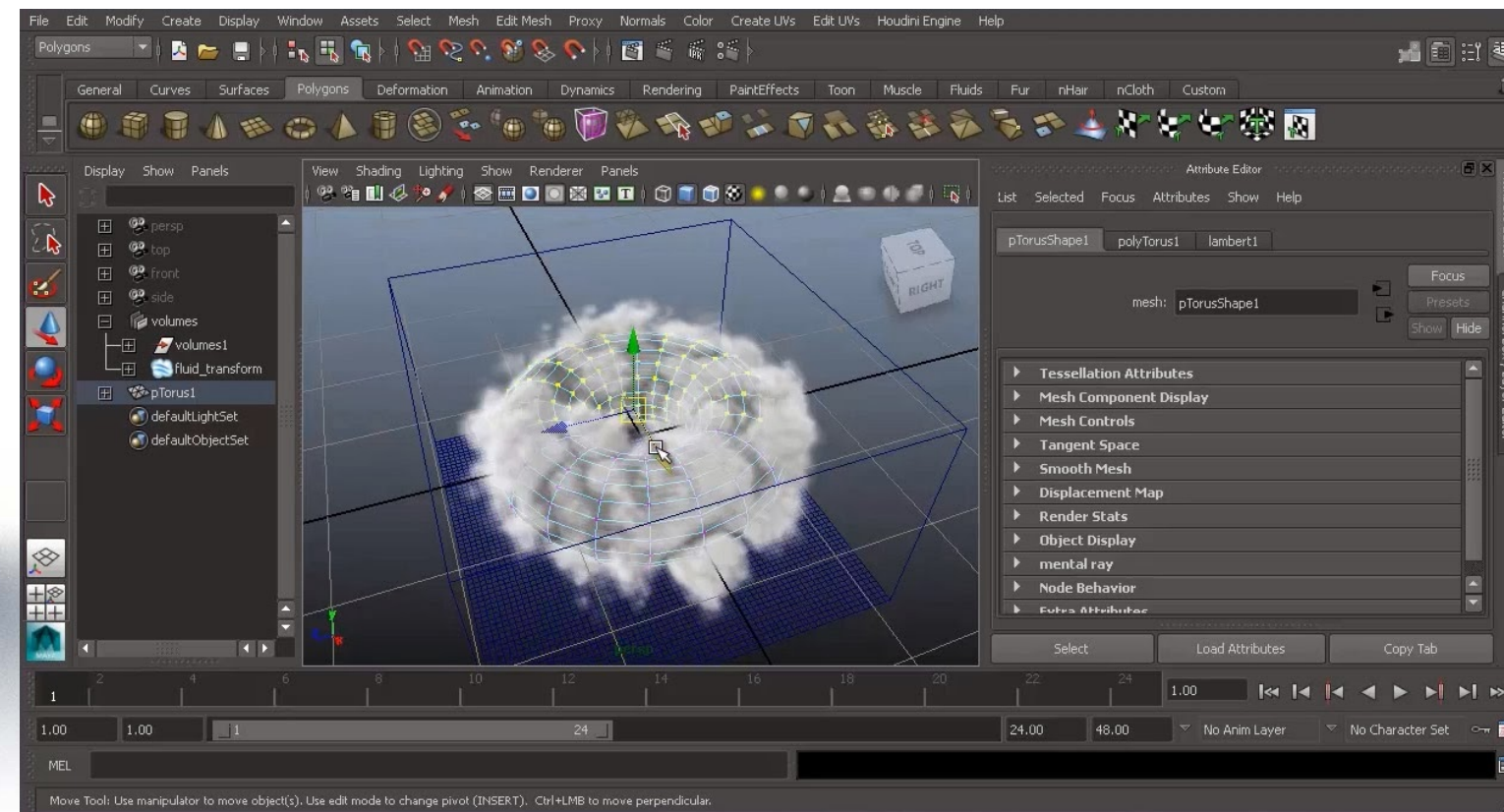
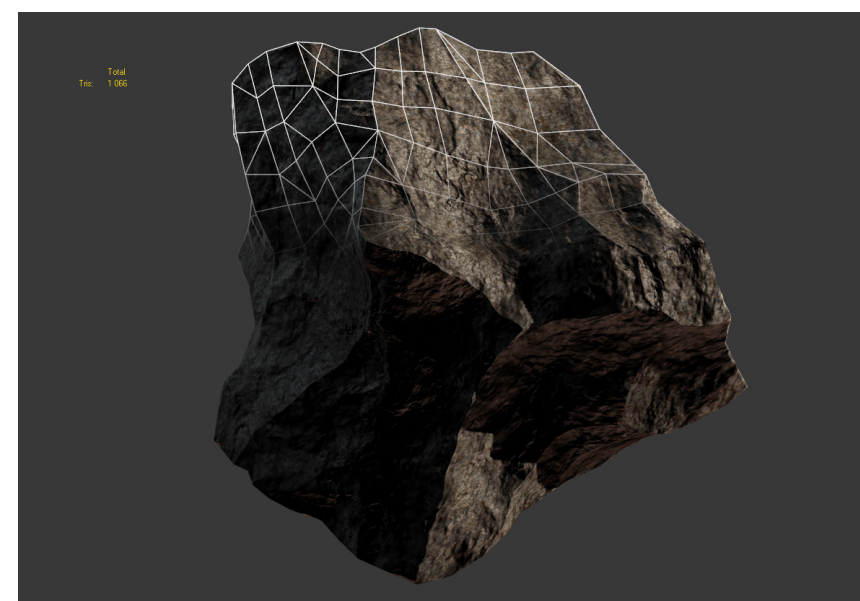
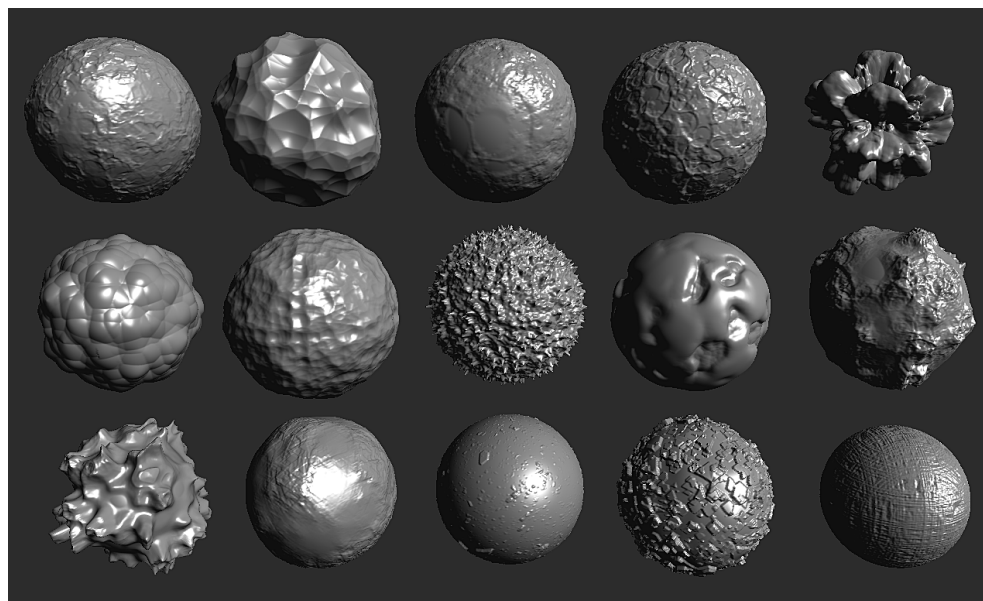
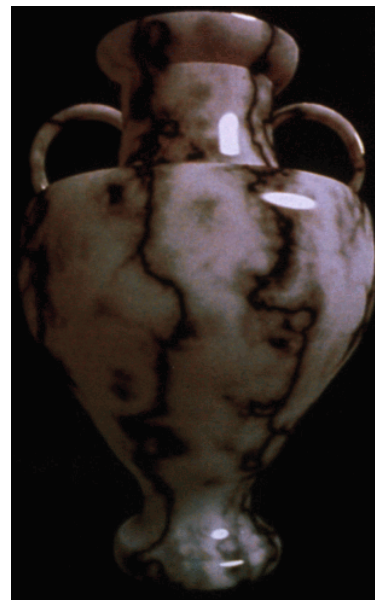
Smooth evolution: $P(x, y, t)$



Perlin noise applications

68/139

In almost every complex shapes ...



Look at [Shader Toy](#) + Noise [example](#)

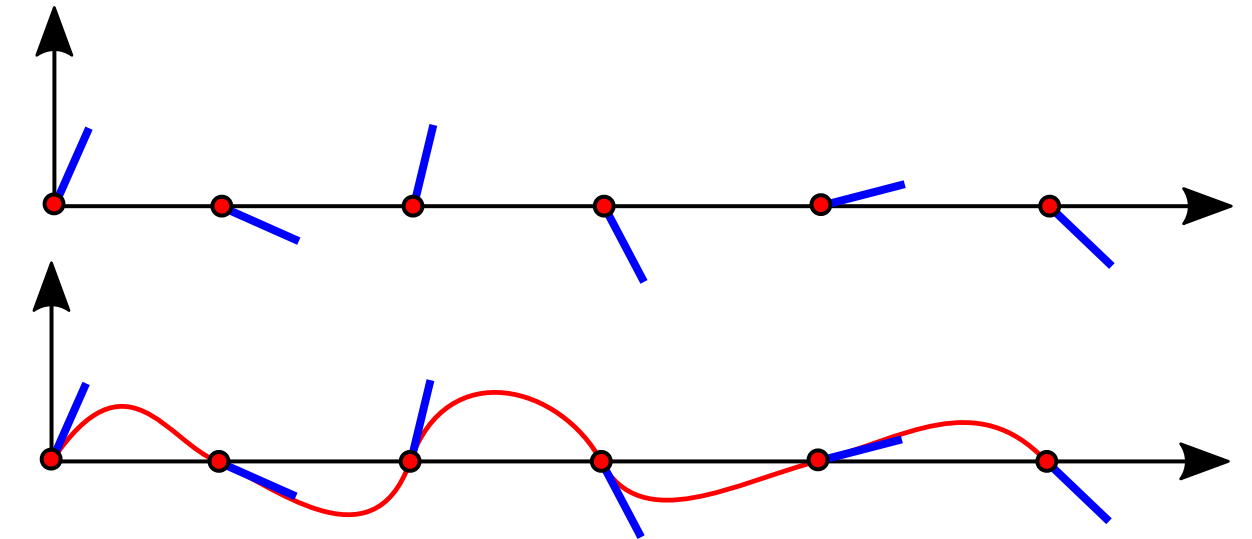
Perlin Noise - Improvement

69/139

Gradient Noise

Use pseudo-random Gradients instead of Positions

Improved frequency control: Oscillate at period 1



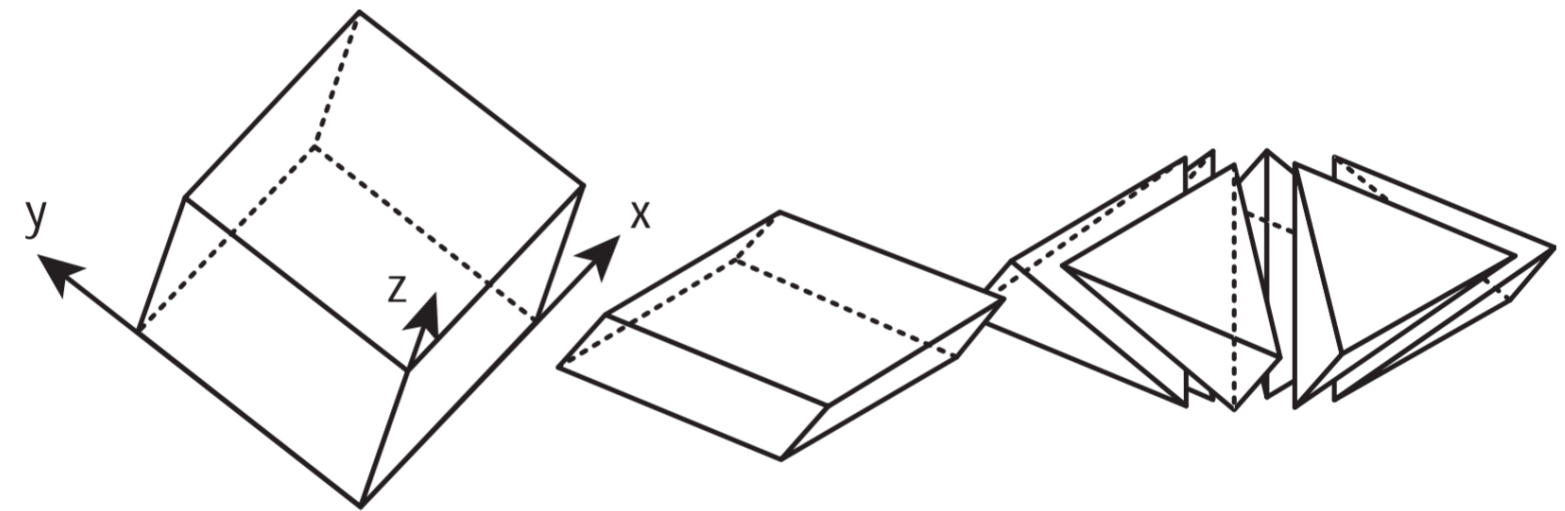
Simplex Noise

Simplex-based interpolation instead of grid interpolation

Avoids grid-direction artifact

Faster for high dimensions

[Stefan Gustavson Simplex noise demystified]



To go beyond:

[A Lagae et al., STAR in Procedural Noise Functions. EG 2010]

Perlin Noise Terrain

70/139

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

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

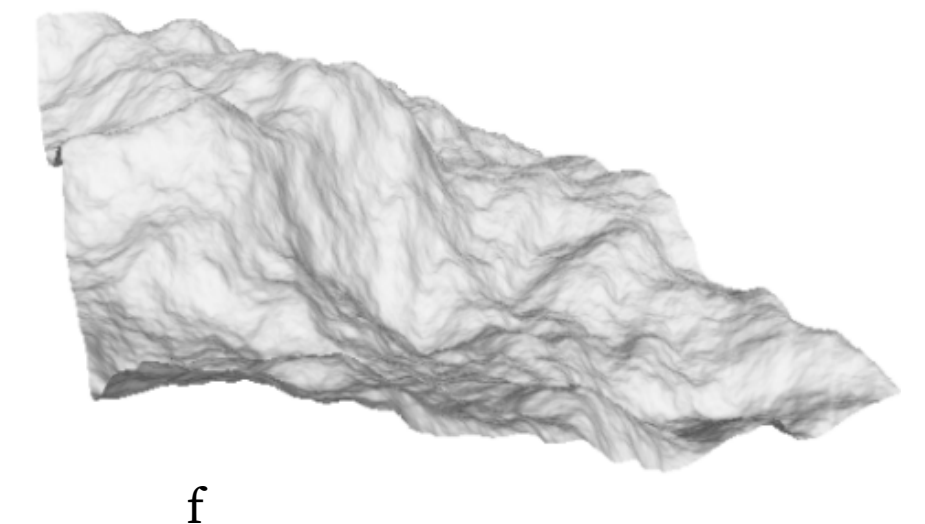
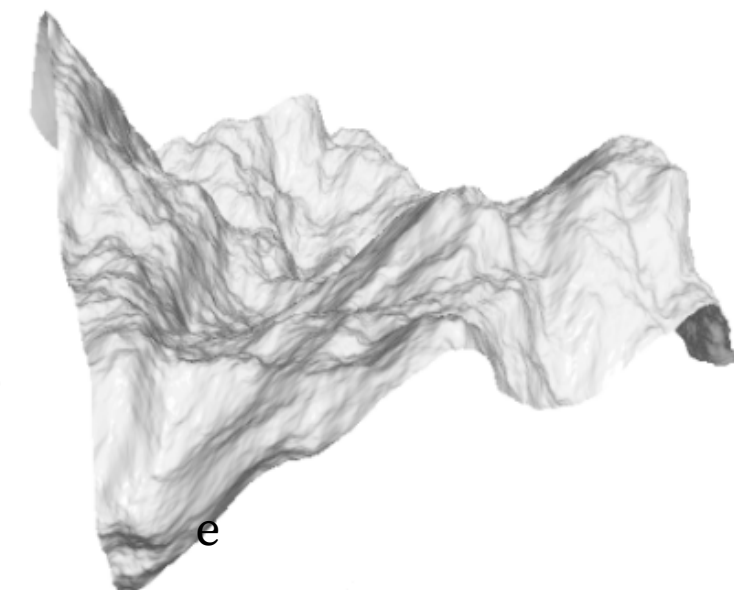
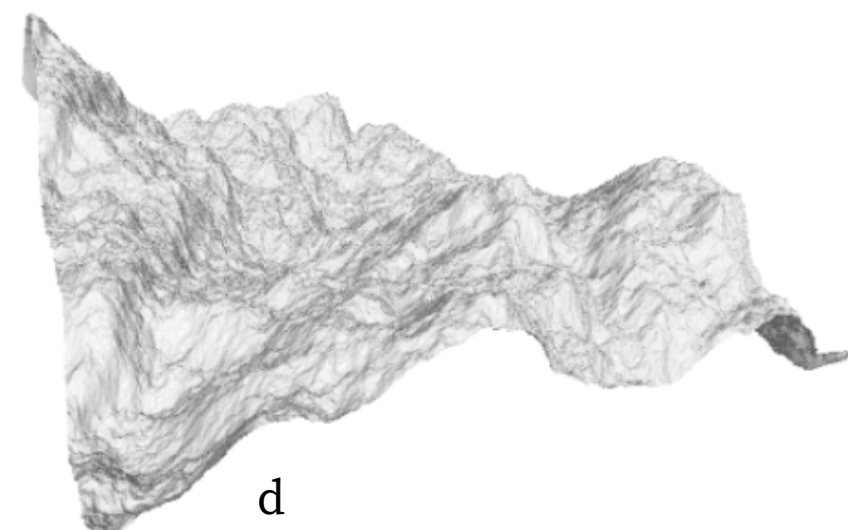
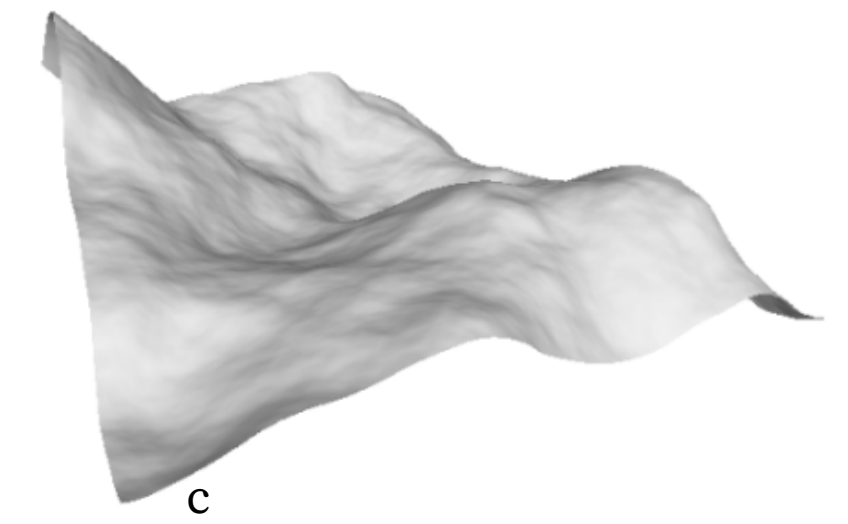
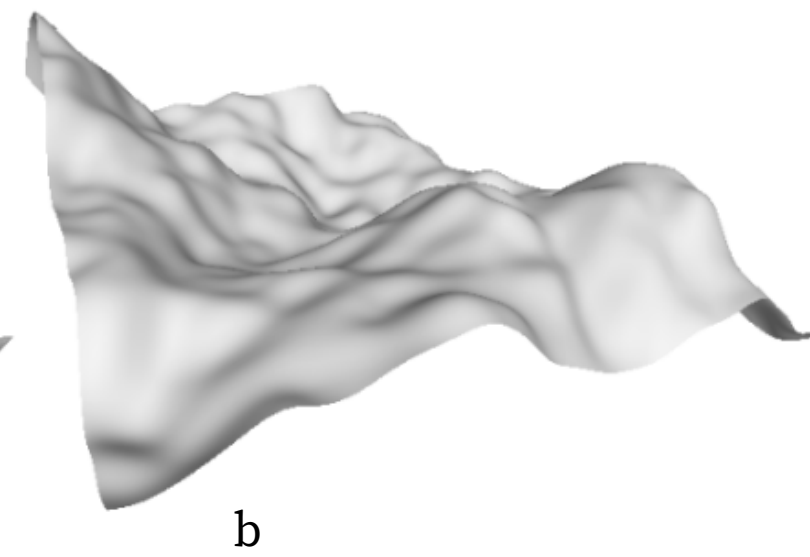
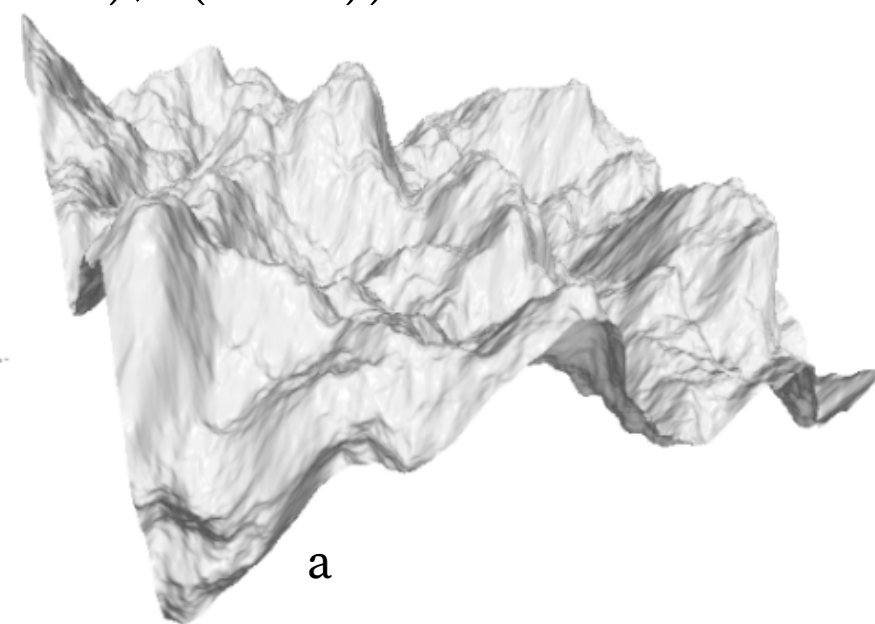
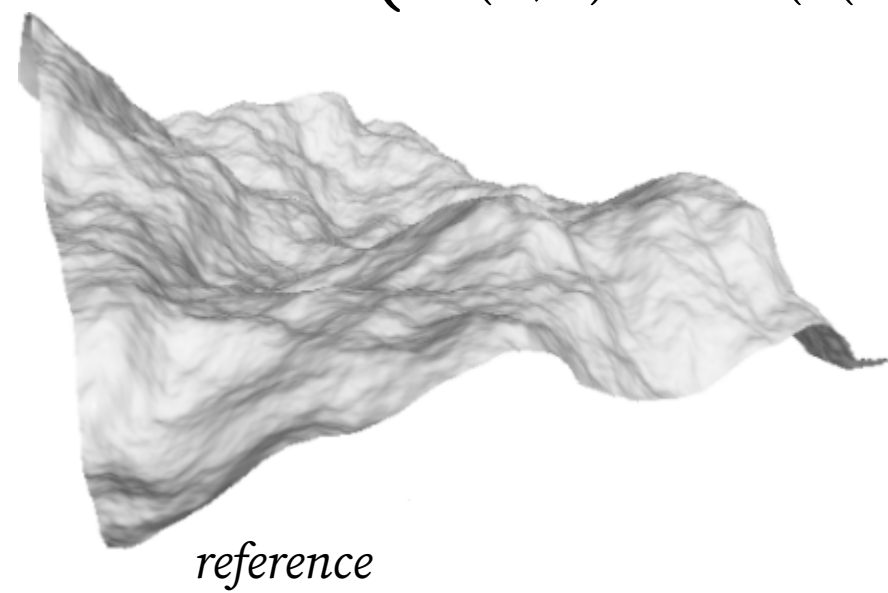
$$N = 9$$

$$\alpha = 0.4$$

$$h = 0.3$$

$$s = 1$$

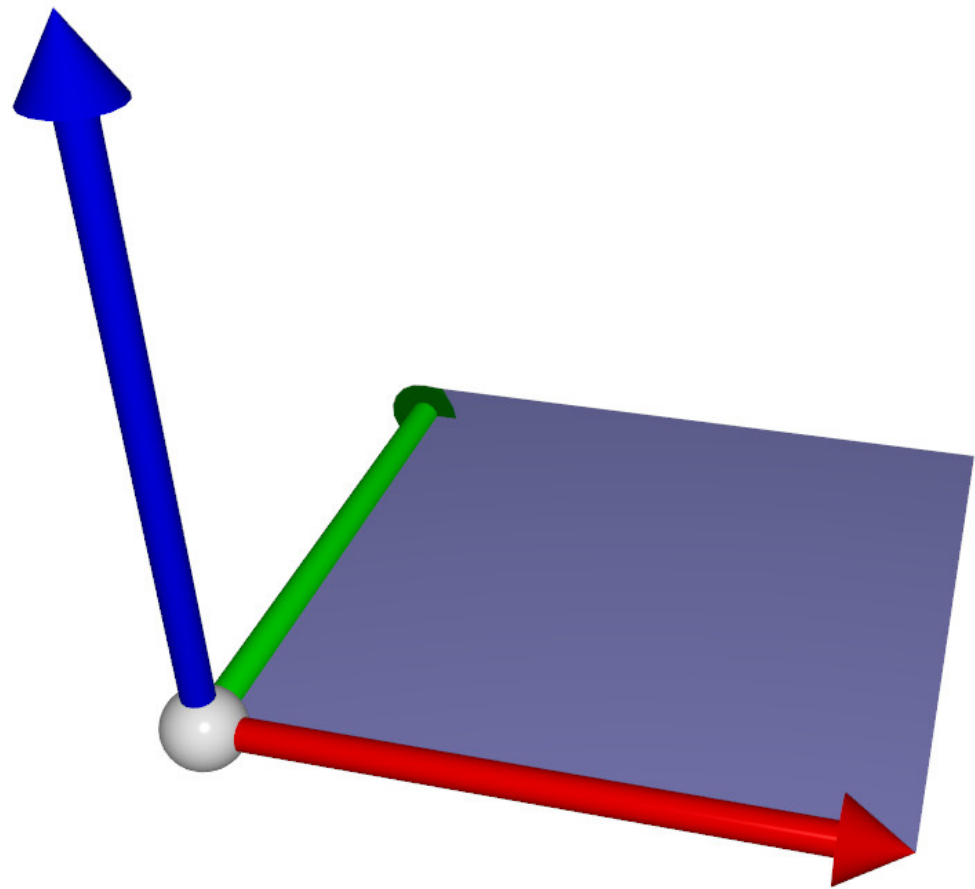
$$o = 0$$



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

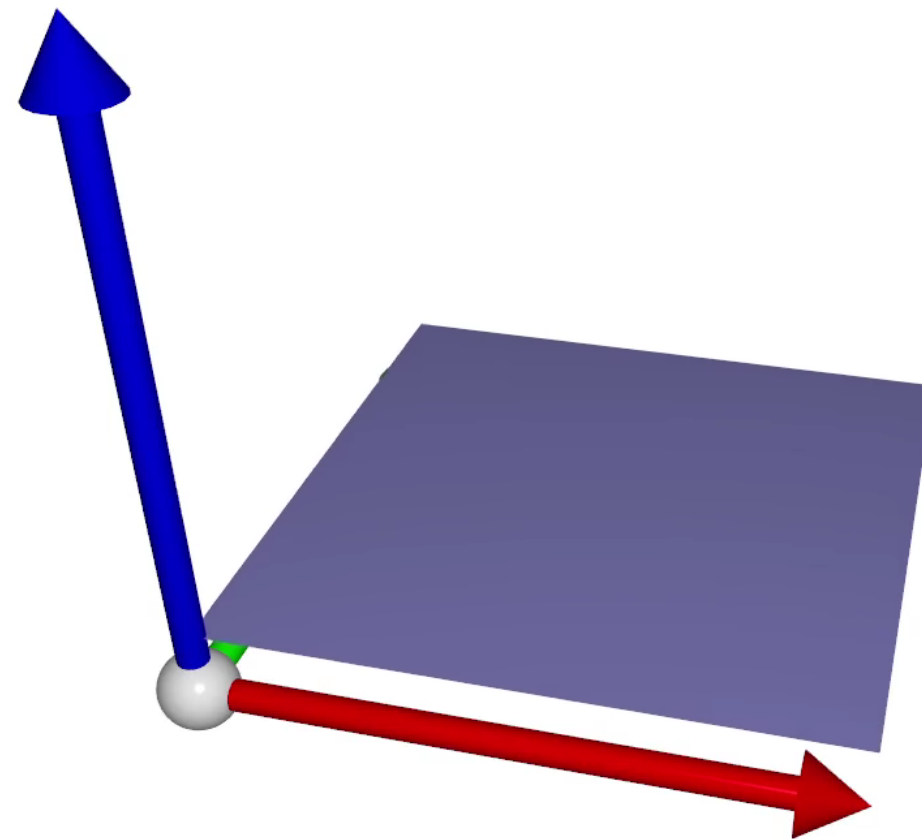
Perlin Noise - Animation

71/139



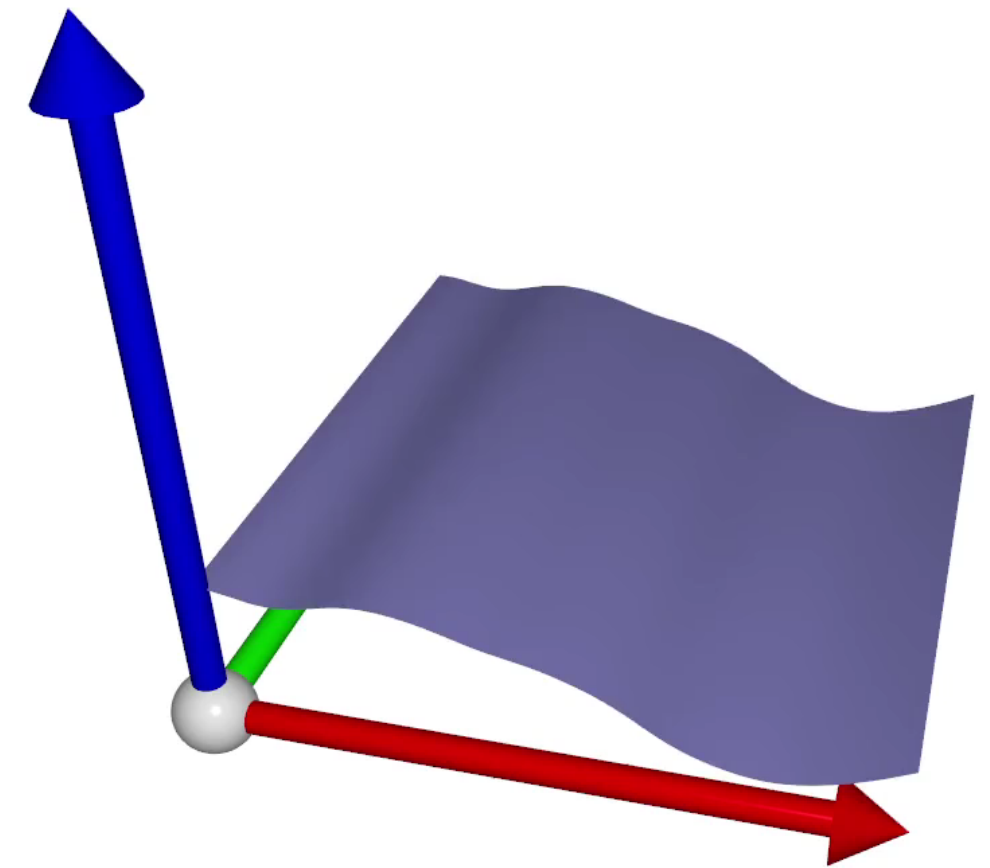
Reference surface function

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



Perlin noise can depends on time

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



Depends on space and time

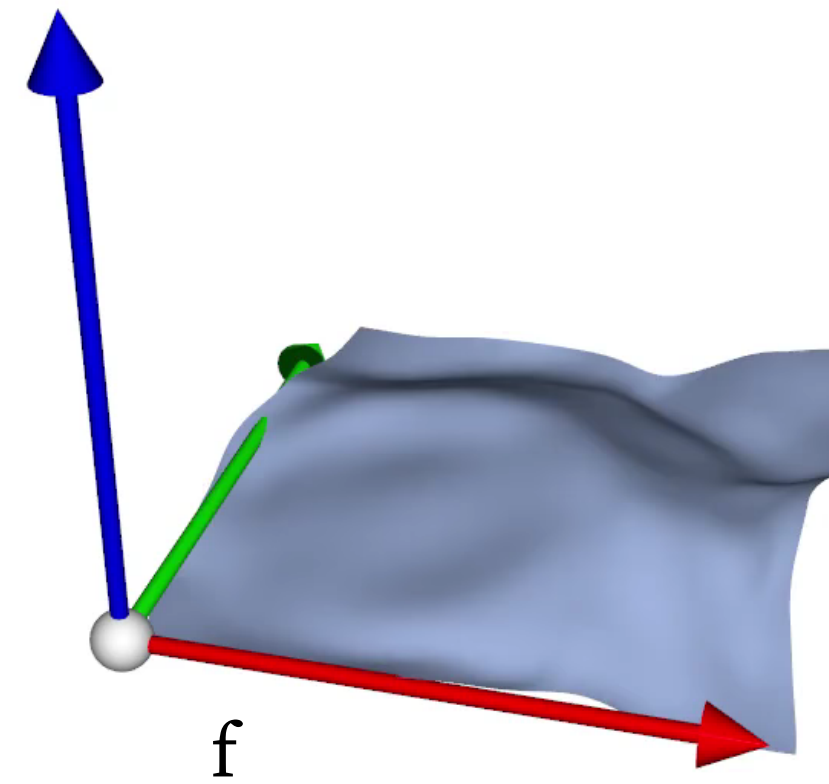
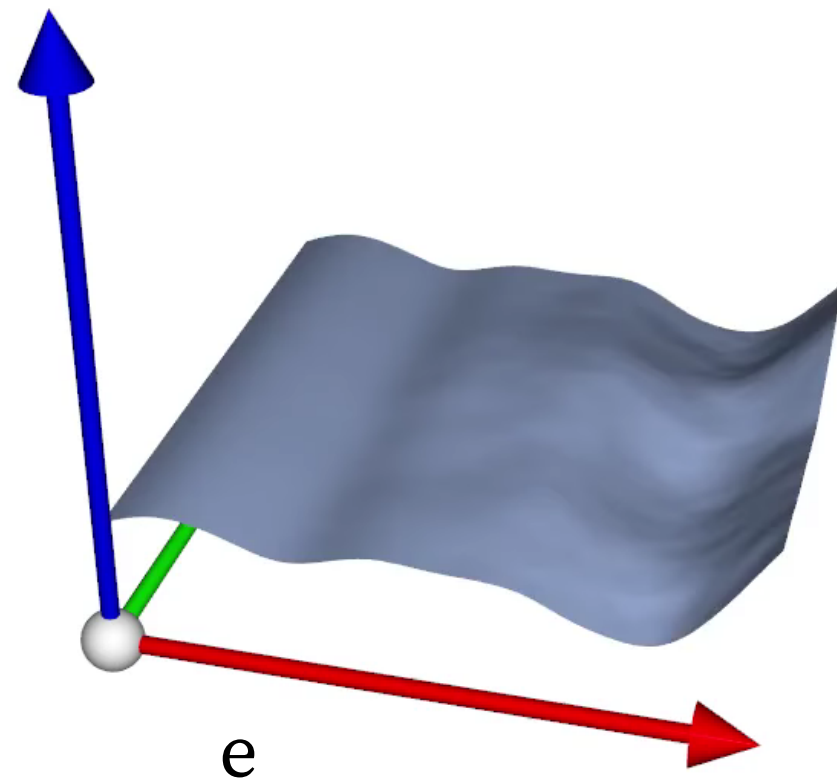
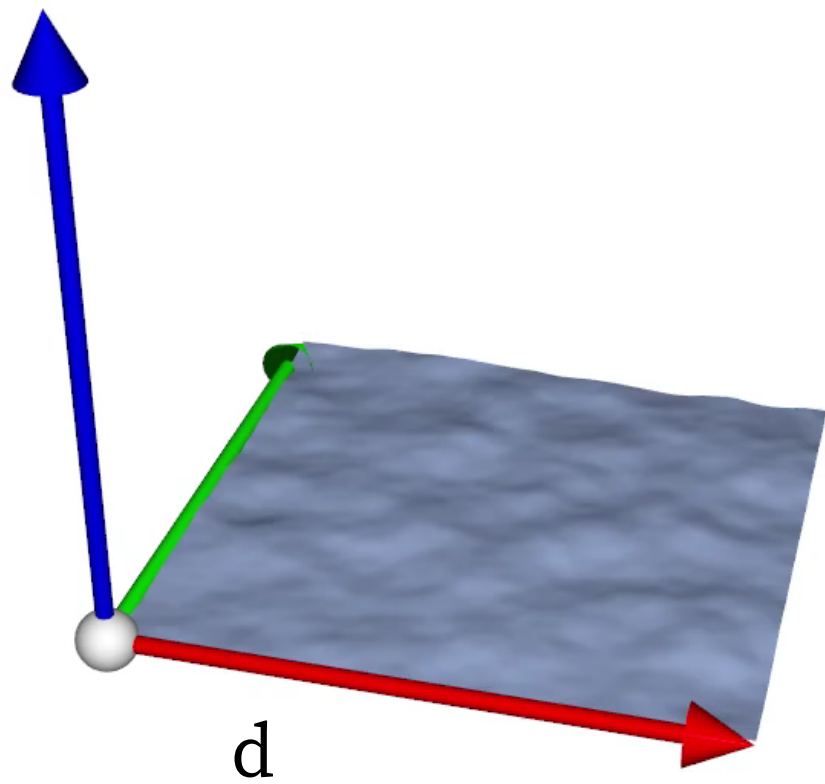
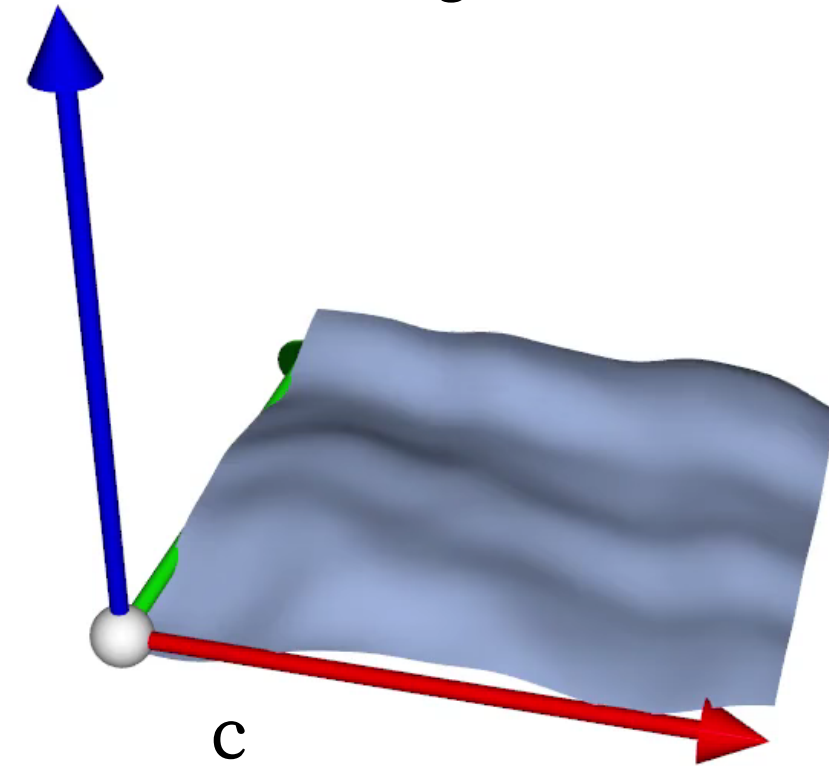
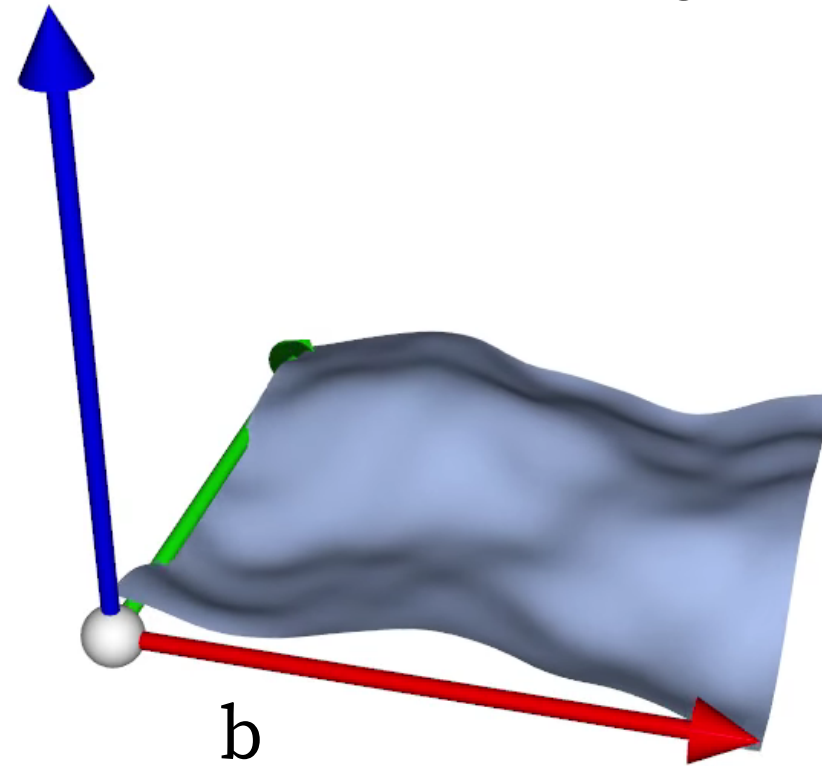
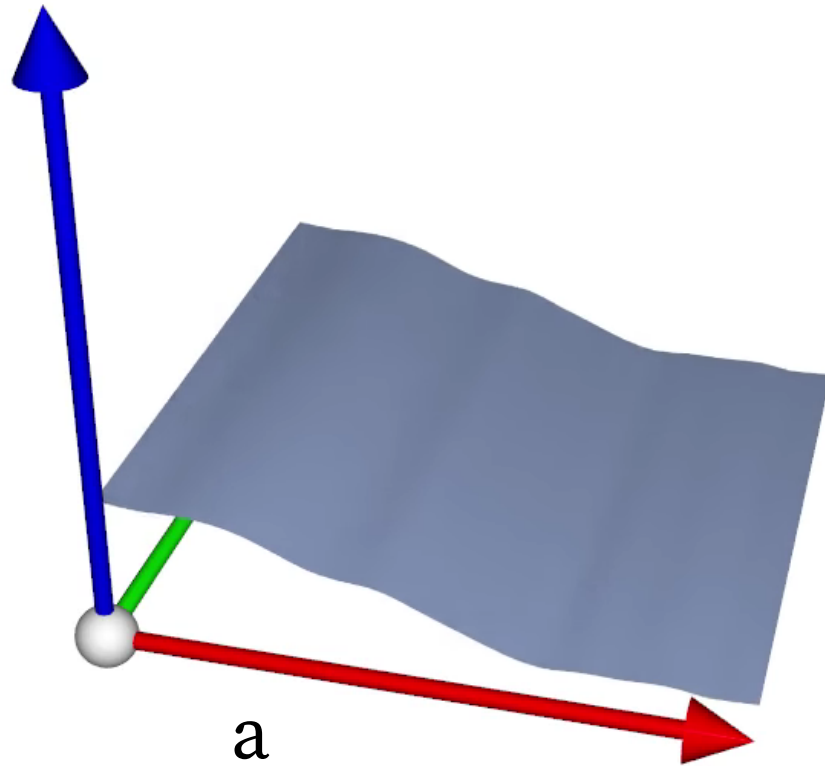
$$S(u, v, t) = (u, v, P(u, t))$$

Perlin Noise - Animation

72/139

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

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

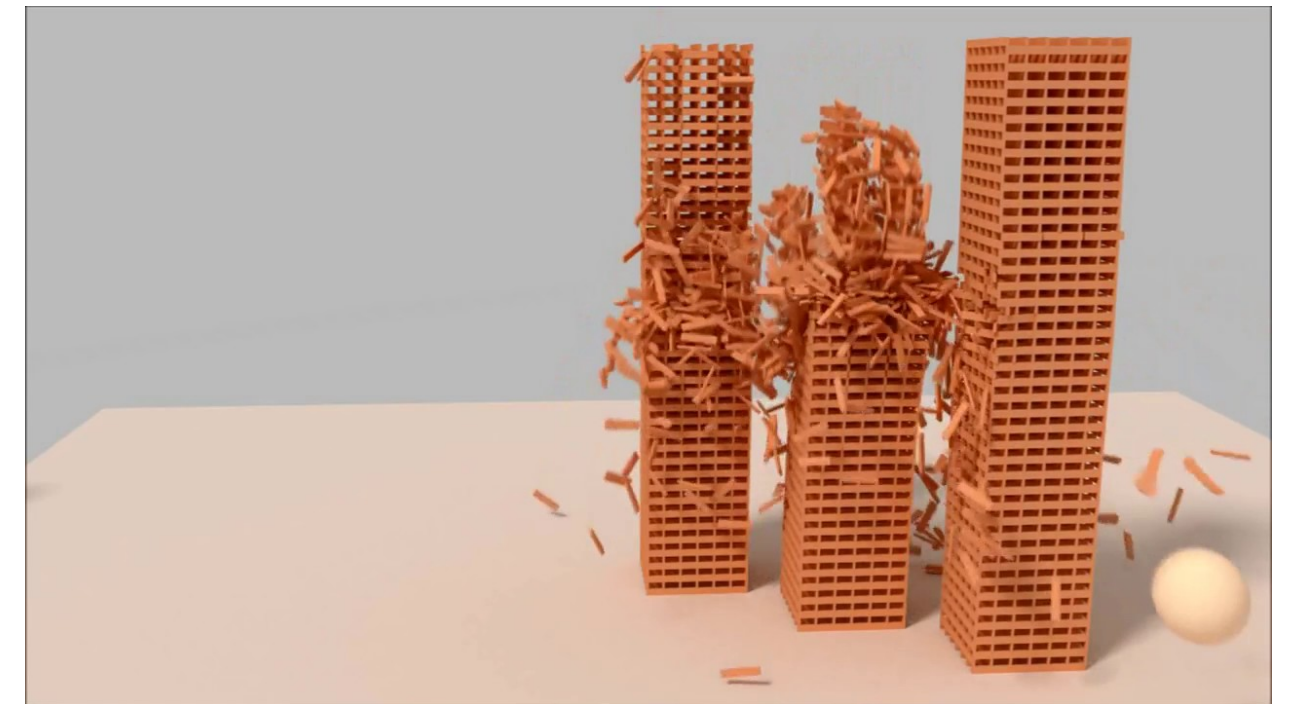
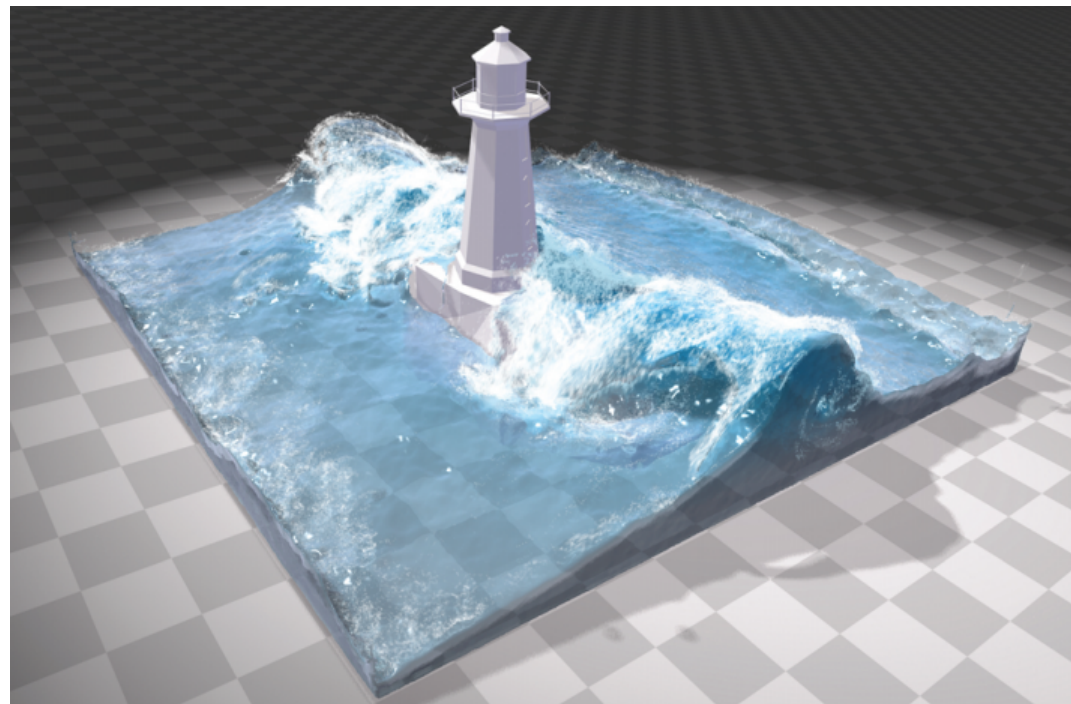


Physically based simulation - Models

When physically based simulation is needed

74/139

- Accurate dynamics
- Tedious to model by hand or procedurally
 - Multiple interacting elements: ex. Multiple collisions: rigid bodies, hairs, etc.
 - Complex animated geometry: Cloths, fluids



1. Description of the system

Describe system by some parameters (positions, speed, orientation, etc).

- State of the system is known at time $t = 0$ - *Initial value problem in time*
- State of the system may be constrained in space - *Boundary value problem in space*

2. Evolution

Link the evolution of the system to forces or constraints using dynamic principles and conservation laws

$\Rightarrow$ Differential equation

3. Numerical Solution

Solve the differential equation using numerical iterative approaches.

Note: Fundamentally different than direct approach controlling the trajectories at key-frames

- The system is set at an initial step
 - We let the numerical solution build the space-time trajectory for us
- (+) Allows to model complex behavior
(-) Lack of control on the result

Fundamental models

1- Particles

2- Rigid bodies

3- Continuum models

Physically-based particle system

77/139

1. Description

Particle is fully described by: Position p , Velocity v , Mass m

Fundamental quantities: position and linear momentum $P = m v$

Linear Momentum preserved in isolated system

2. Evolution

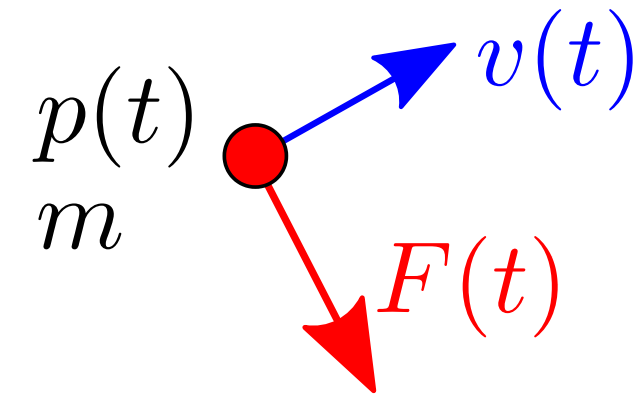
- Fundamental principle of dynamics

Force applied on particle $F(p, v, t)$

$$\begin{cases} p'(t) = v(t) \\ P'(t) = m v'(t) = F(p, v, t) \end{cases}$$

- Conservation of energy (ex. kinetic energy $(1/2 m v^2)$ +potential energy = const, etc.)

- Lagrangian, or Hamiltonian (reduced coordinates)



Physically-based particle system

78/139

3. Numerical Solution

ODE (Ordinary Differential Equation) formulated as an **Initial Value Problem**

$$\text{ex. } \begin{cases} p'(t) = v(t) \\ mv'(t) = F(p, v, t) \end{cases}, \quad \text{with } v(0) = v_0, p(0) = p_0$$

- Discretize in time $t^k = k h$, $h = \Delta t = \text{time step}$.
 $\Rightarrow$ Build a discrete numerical solution $p^k = p(t^k)$, $v^k = v(t^k)$.

- We can consider initially the following iterative scheme

$$\begin{cases} v^{k+1} = v^k + h F(p^k, v^k, t^k) \\ p^{k+1} = p^k + h v^{k+1} \end{cases}$$

Simple to implement, reasonably OK for simple examples (more details later).

Physically-based particle system

79/139

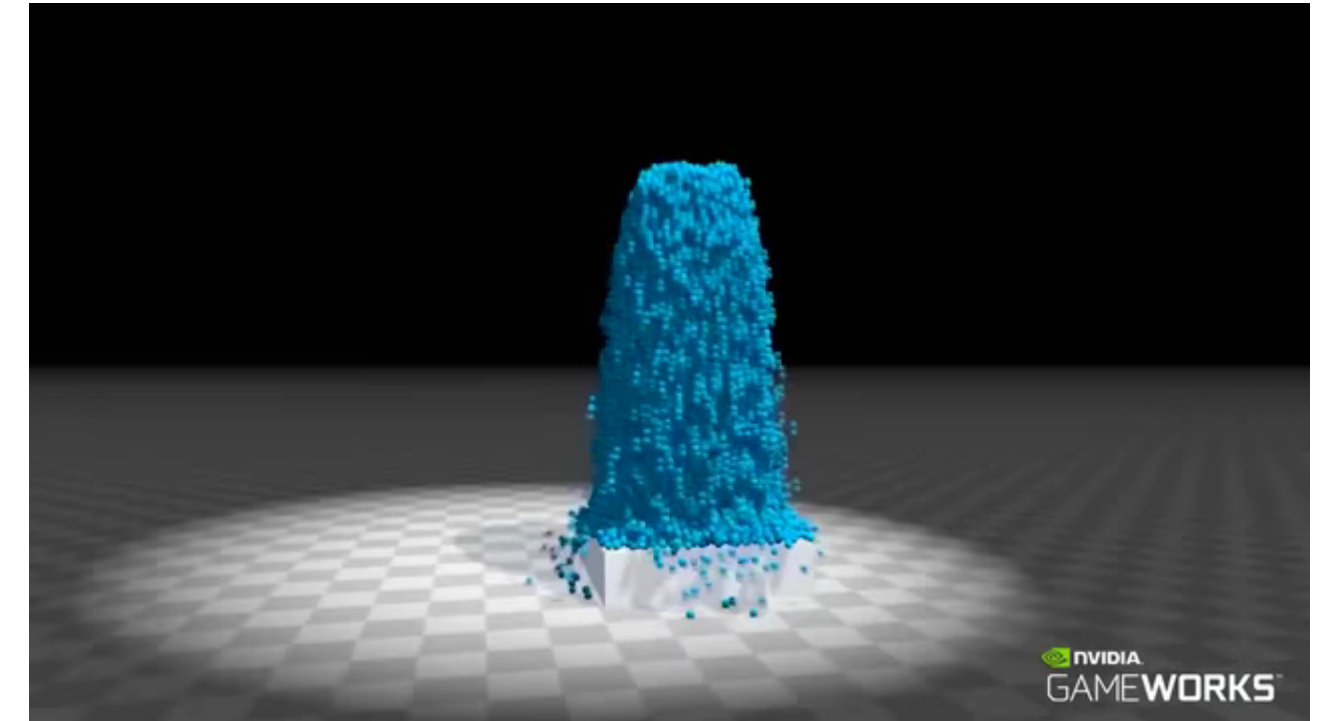
Pro

- (+) Simple to implement, and control.
- (+) Efficient to compute, scalable
- (+) Highly adaptable from simple particle to rigid and deformable models

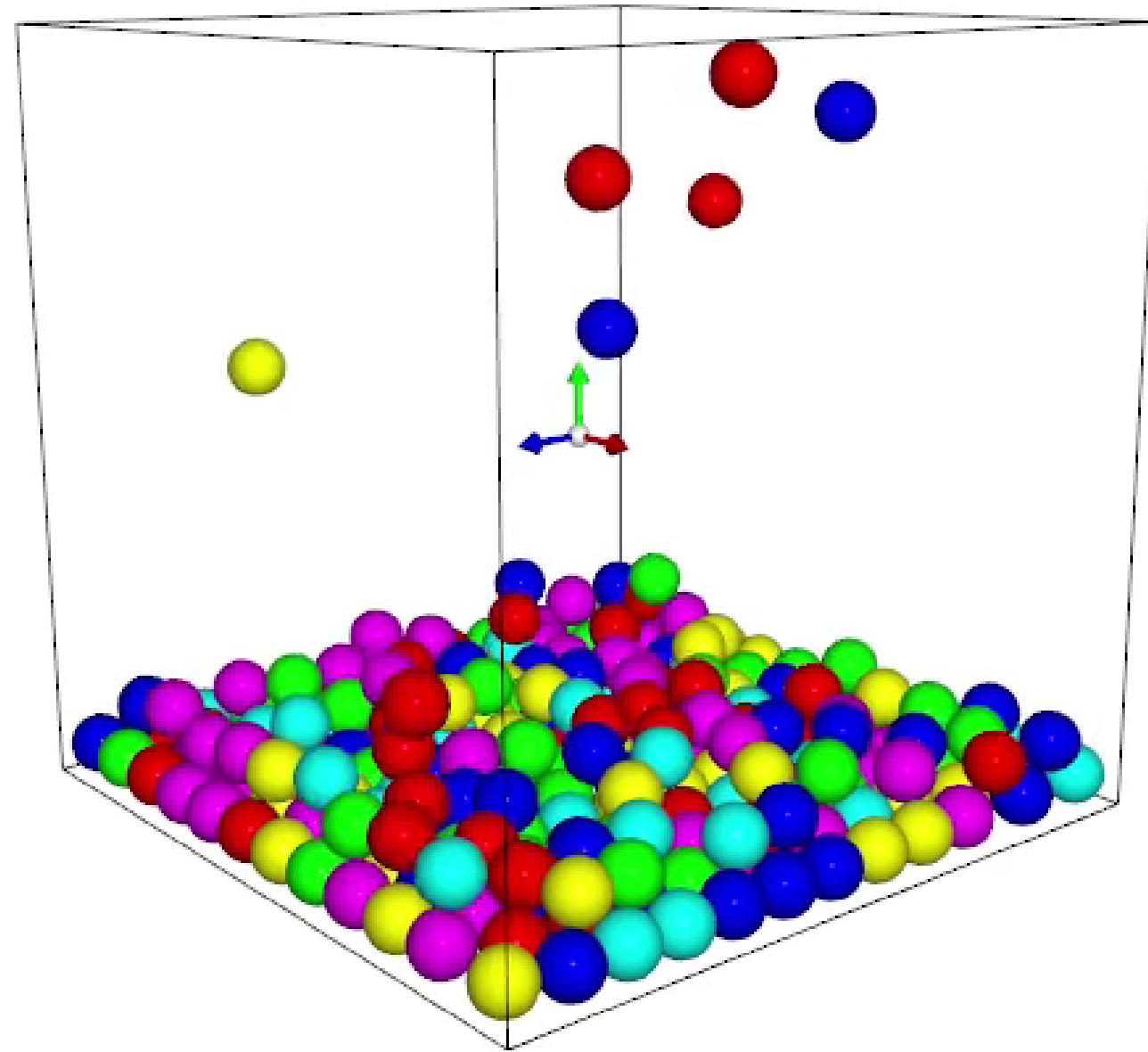
Cons

- (-) Limited accuracy - highly simplified model from physical point of view.
⇒ Dominant model in CG production for general purpose deformable model animation.

Common use: Lots of sparsely interacting particles



Rigid spheres

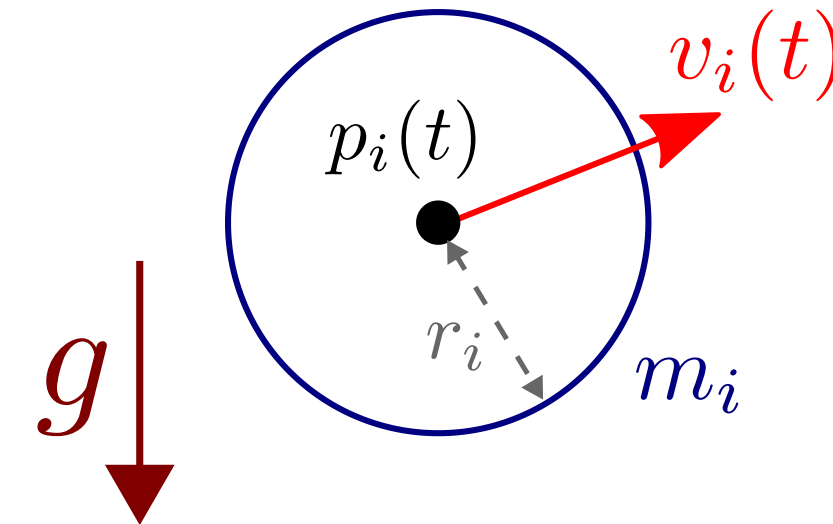


System modeling

81/139

Particles modeling the center of hard spheres.

- Spheres can collide with surrounding obstacles
- Spheres can collide with each others
- *System*: N particles with position p_i , velocity v_i , mass m_i , modeling a sphere of radius r_i .
 - Initial conditions $p_i(0) = p_i^0, v_i(0) = v_i^0$
- *Forces*: Single gravity forces $F_i = m_i g$. Collisions handled by *impulses*.
- *Temporal evolution*: Fundamental principle of dynamics $v_i(t) = p'_i(t), v'_i(t) = g$
- *Numerical solution*
$$\begin{cases} v^{k+1} = v^k + h g \\ p^{k+1} = p^k + h v^{k+1} \end{cases}$$



Collision with a plane

82/139

Plane $\mathcal{P}$: parameterized using a point a and its normal n .

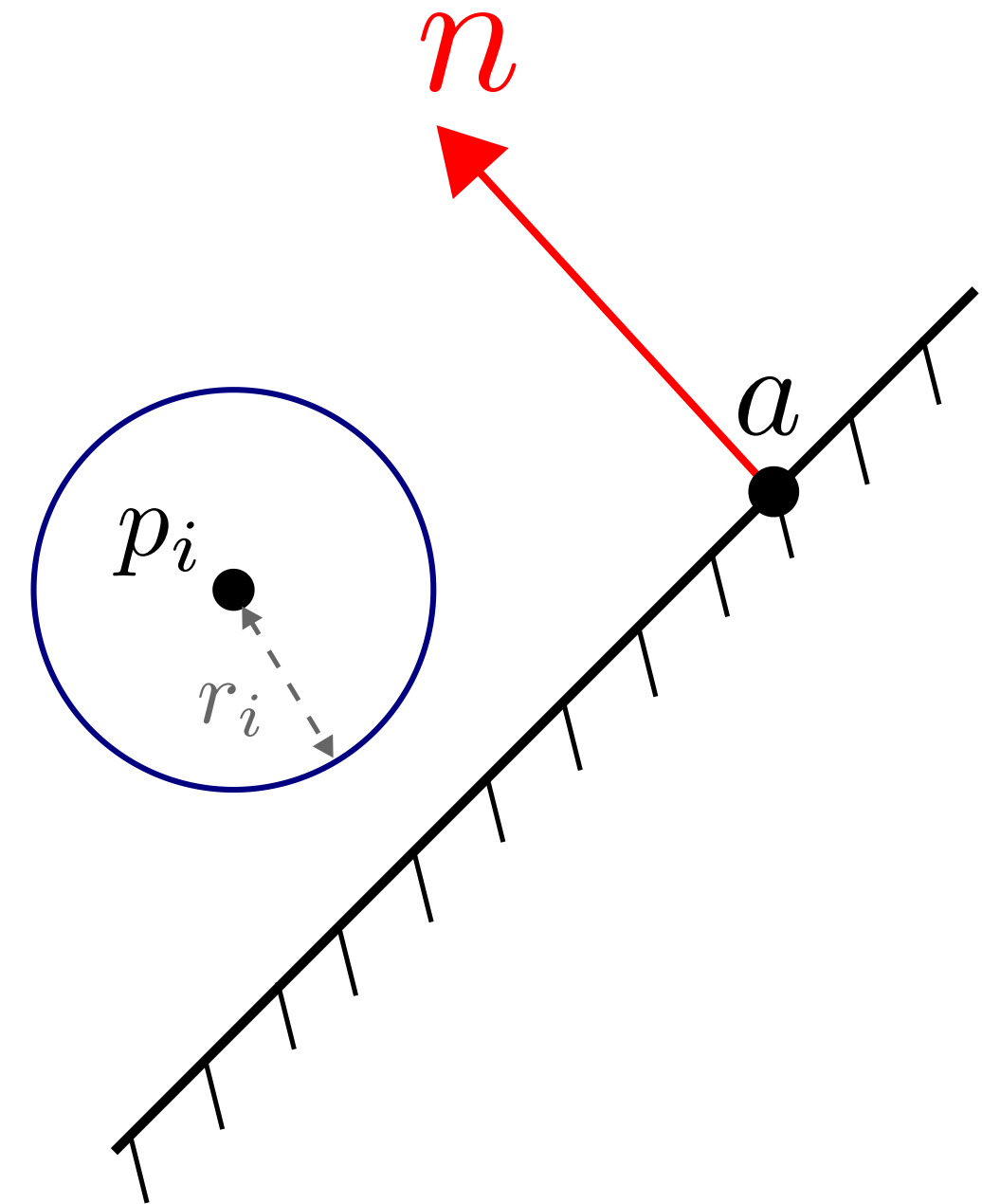
$$\{p \in \mathbb{R}^3 \in \mathcal{P} \Rightarrow (p - a) \cdot n = 0\}$$

- Sphere above plane : $(p_i - a) \cdot n > r_i$
- Sphere in collision: $(p_i - a) \cdot n \leq r_i$

- Collision detection algorithm

```
for(int i=0; i<N; ++i)
{
    float detection = dot(p[i]-a, n);
    if (detection <= r[i])
    {
        // ... collision response
    }
}
```

What should we do when a collision is detected



Collision response with plane

83/139

Suppose exact contact: $(p_i - a) \cdot n = r_i$

Collision response = **Update velocity**

Split $v = v_{//} + v_{\perp}$

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

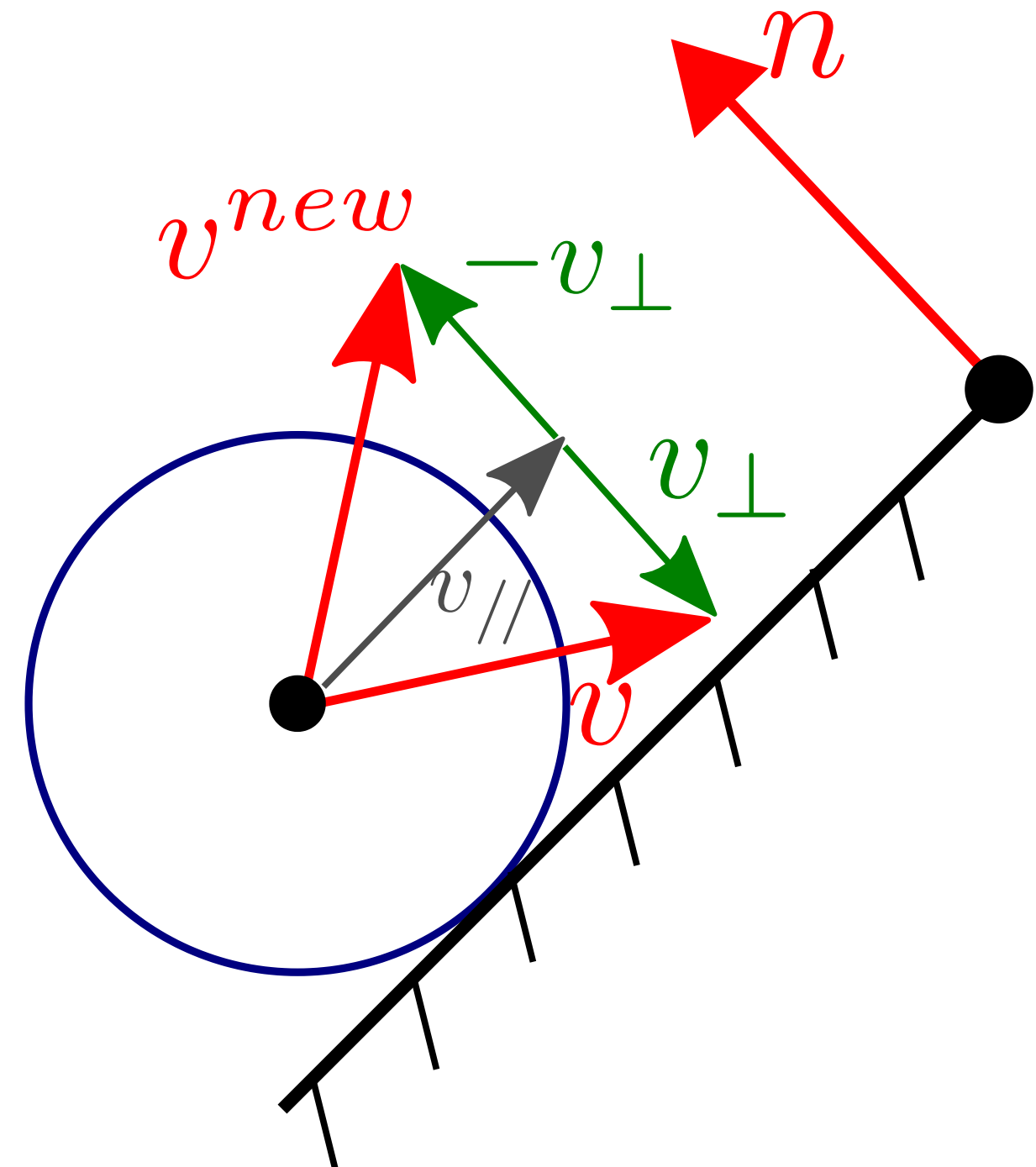
$$-v_{//} = v - (v \cdot n)n$$

New velocity

$$v^{new} = \alpha v_{//} - \beta v_{\perp}$$

$\alpha \in [0, 1]$ Restitution coefficient in $//$ direction (friction)

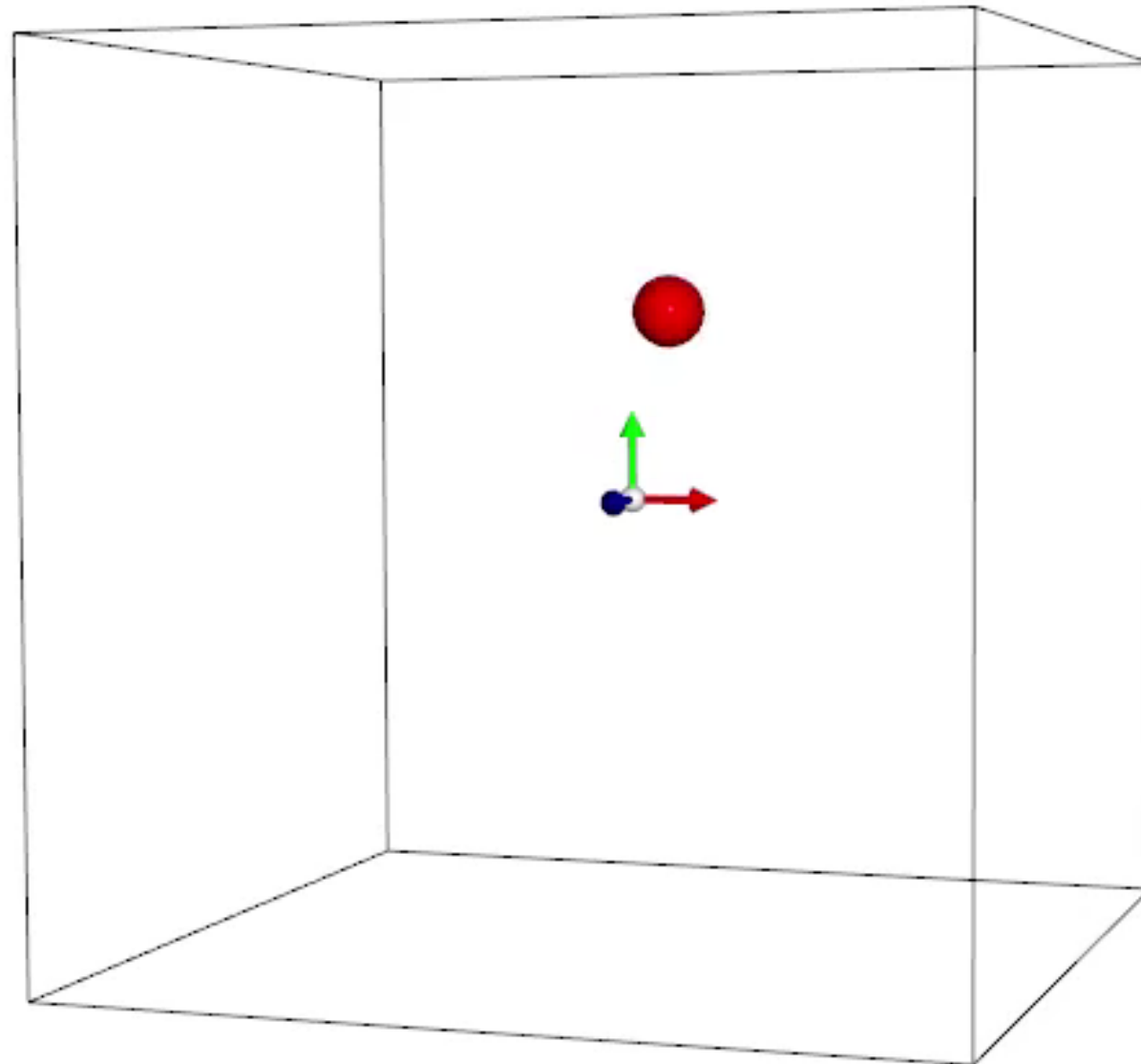
$\beta \in [0, 1]$ Restitution coefficient in $\perp$ direction (impact)



Result: Collision response

84/139

Applying collision response on speed only

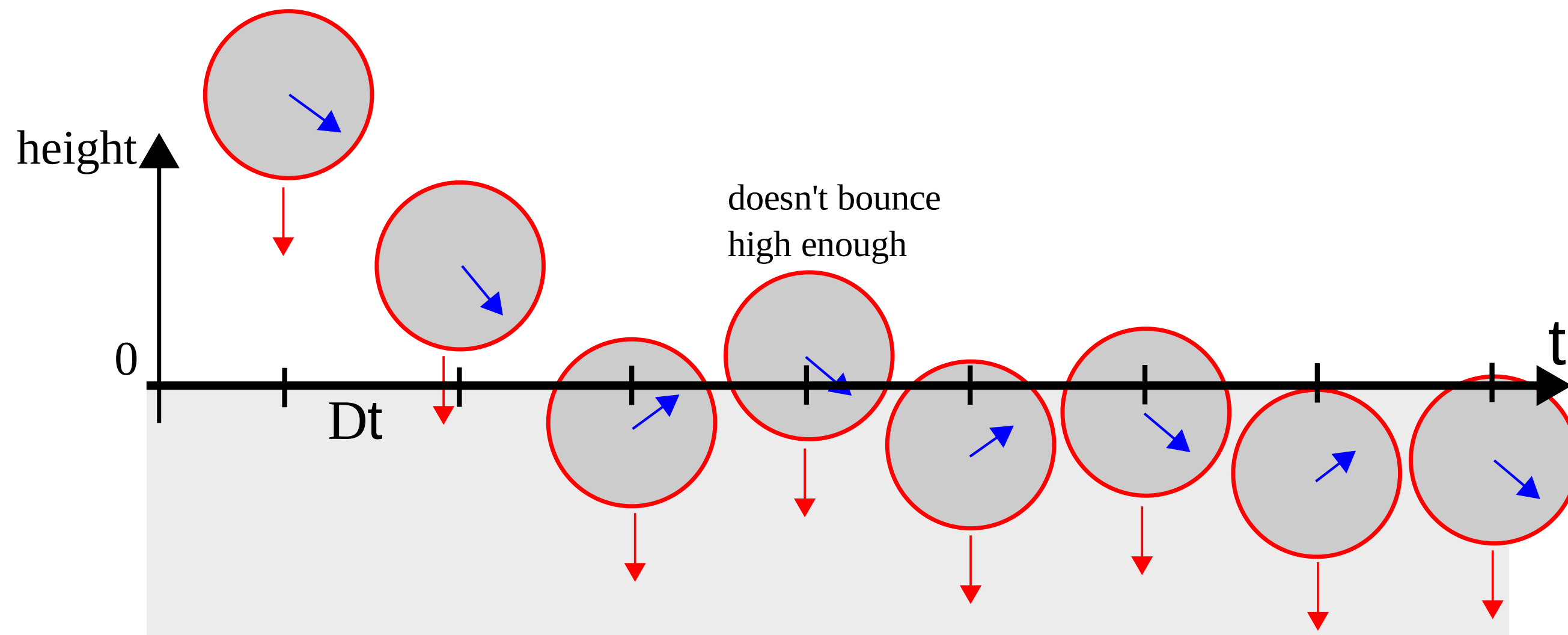


Result: Collision response - issue with discrete time ^{85/139}

We assumed contact b/w sphere and plane

But: Exact contact never happens in discrete time

- When collision is detected $\rightarrow$ already inside the wall
- Weight is still acting



Collision response with plane : position

86/139

In real case (discrete time) no exact contact, but penetration $(p_i - a) \cdot n_i < r_i$
 $\Rightarrow$ Need to compute collision response at contact point.

Three possibilities

(1) Correct position in projecting on the
constraint

(+) *Simple to implement*

(-) *Physically incorrect position*

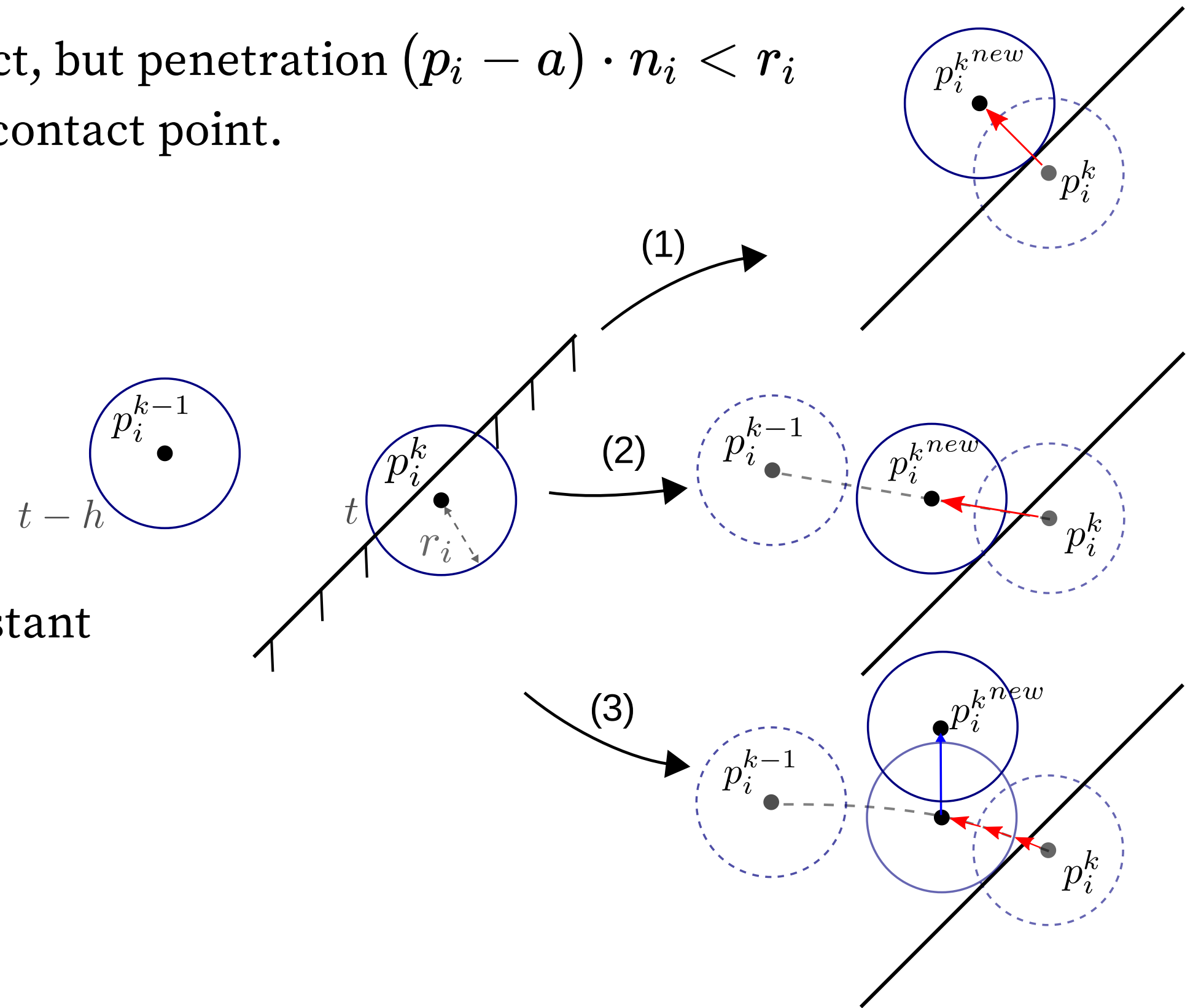
(2) Approximate the correct position

(3) Go backward in time to find exact instant
of collision

Continuous Collision Detection

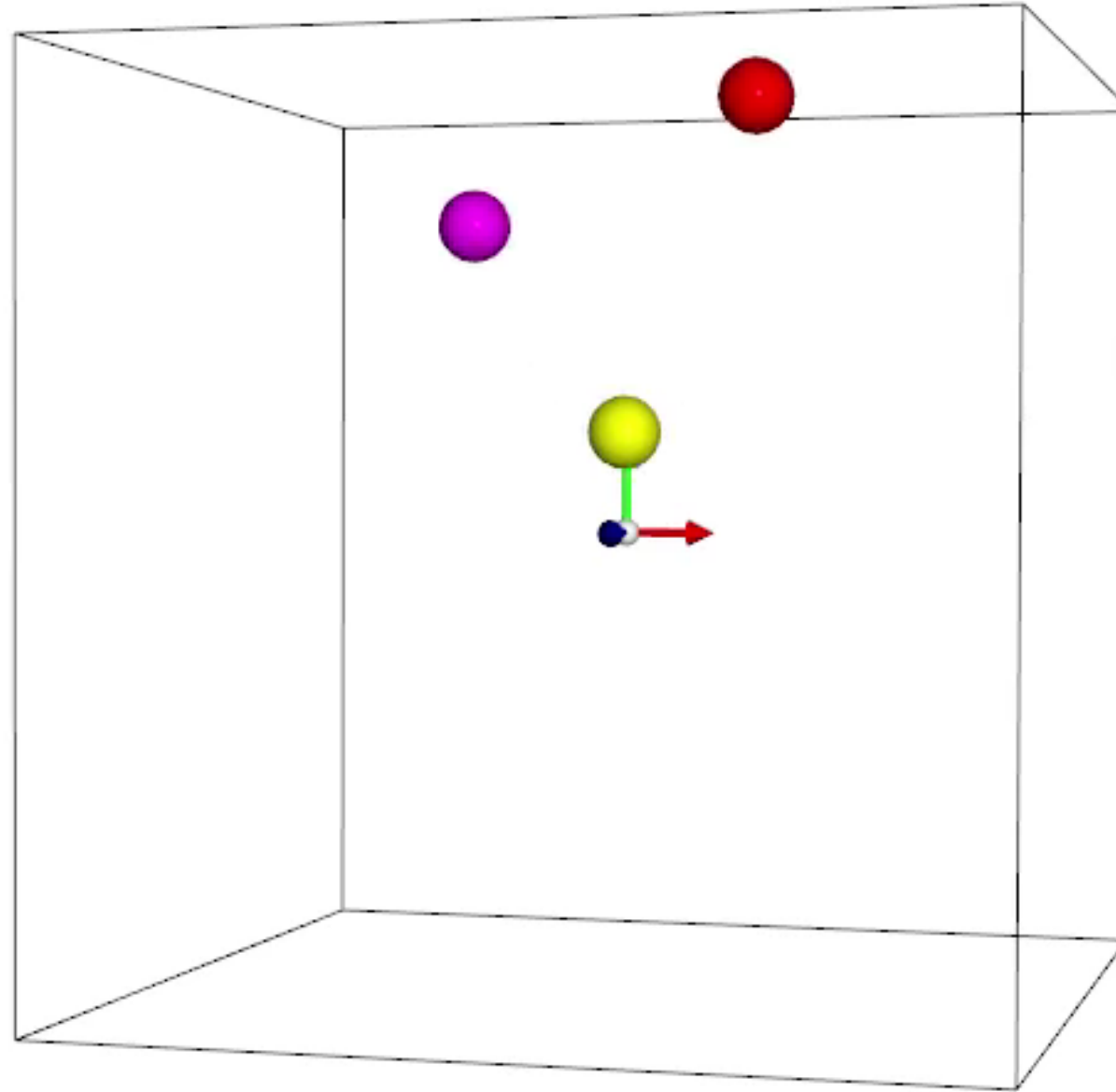
(+) *Physically correct*

(-) *Computationally heavy (binary search, etc.)*



Result: Projecting position on plane

87/139



$$p_i^{new} = p_i + d n$$

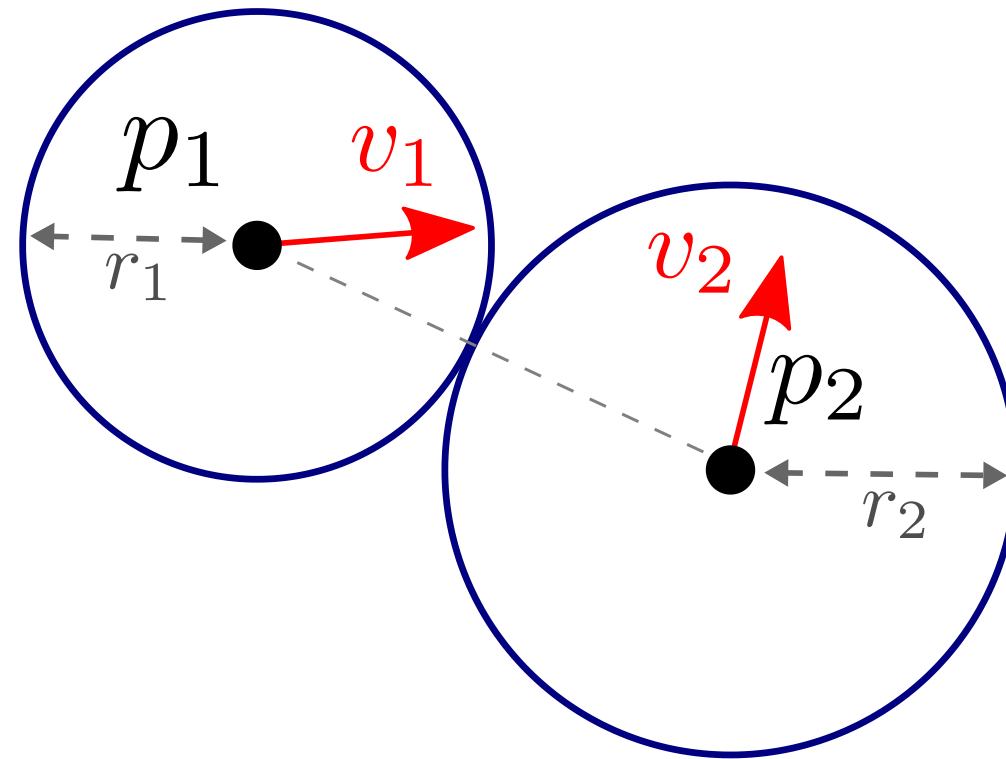
$$d = r_i - (p_i - a) \cdot n_i : \text{distance of penetration}$$

Collision between spheres

88/139

Given 2 spheres $(p_1, v_1, r_1, m_1), (p_2, v_2, r_2, m_2)$.

Collision when $\|p_1 - p_2\| \leq r_1 + r_2$



What happen with their velocities ?

$$v_1 \rightarrow v_1^{new}, v_2 \rightarrow v_2^{new}$$

Notion of impulse

89/139

An impulse J is the integrated force over time $J = \int_{t_1}^{t_2} F(t) dt$

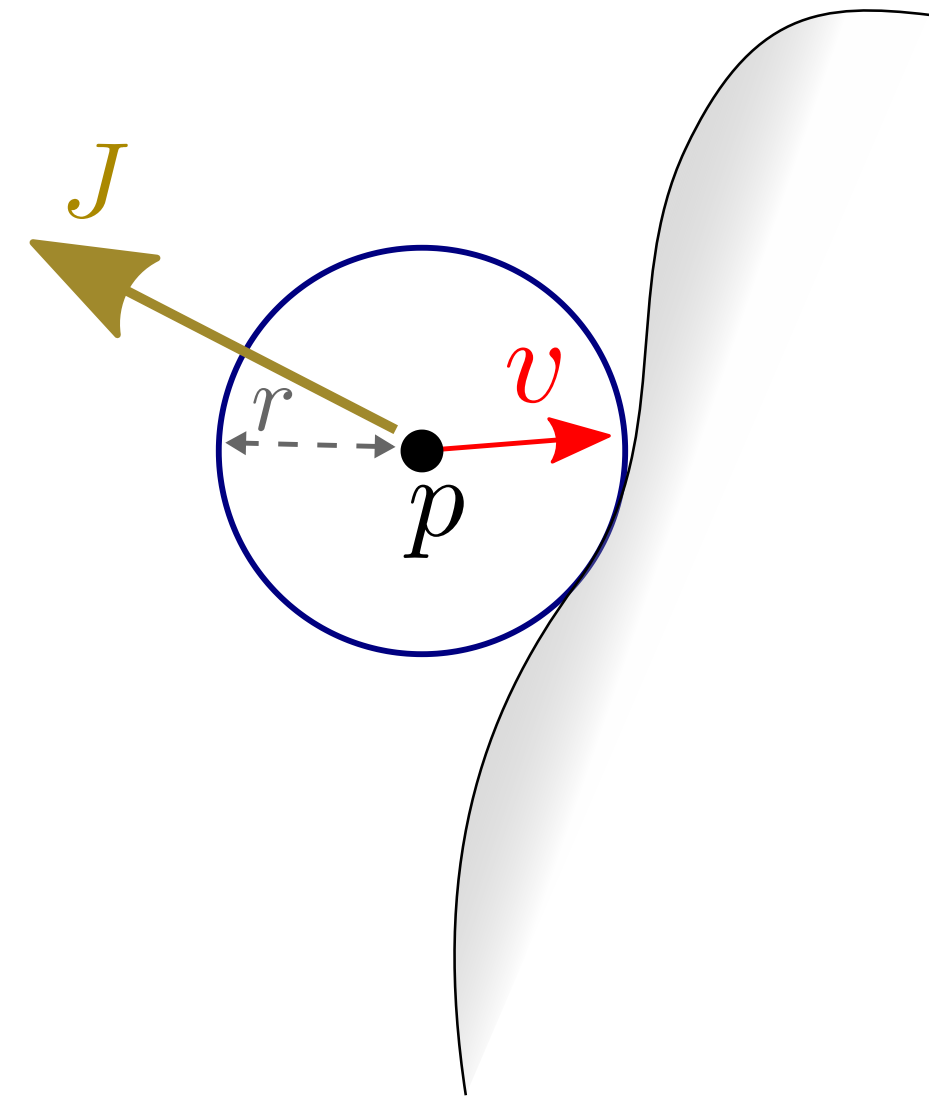
→ results in a sudden change of speed (/momentum) in a discrete case

For a particle with constant mass

$$\int_{t_1}^{t_2} F(t) dt = \int_{t_1}^{t_2} m a(t) dt$$
$$\Rightarrow J = m (v(t_2) - v(t_1))$$

For an impact $v \rightarrow v^{new}$

$$v^{new} = v + J/m$$



Two spheres in collision

90/139

Impulse orthogonal to the separating plane between the two surfaces

$$J = j u, \quad u = (p_1 - p_2) / \|p_1 - p_2\|$$

The system with the two spheres is preserving its linear momentum

$\Rightarrow$ Respective impulses j are equals in magnitude, and opposed in direction

$$m_1 v_1 + m_2 v_2 = m_1 v_1^{new} + m_2 v_2^{new} \Rightarrow m_1 (v_1^{new} - v_1) = -m_2 (v_2^{new} - v_2) \Rightarrow J_1 = -J_2$$

Assume collision of "hard spheres" = "Elastic collision"

= No loss of energy, conservation of kinetic energy of the system

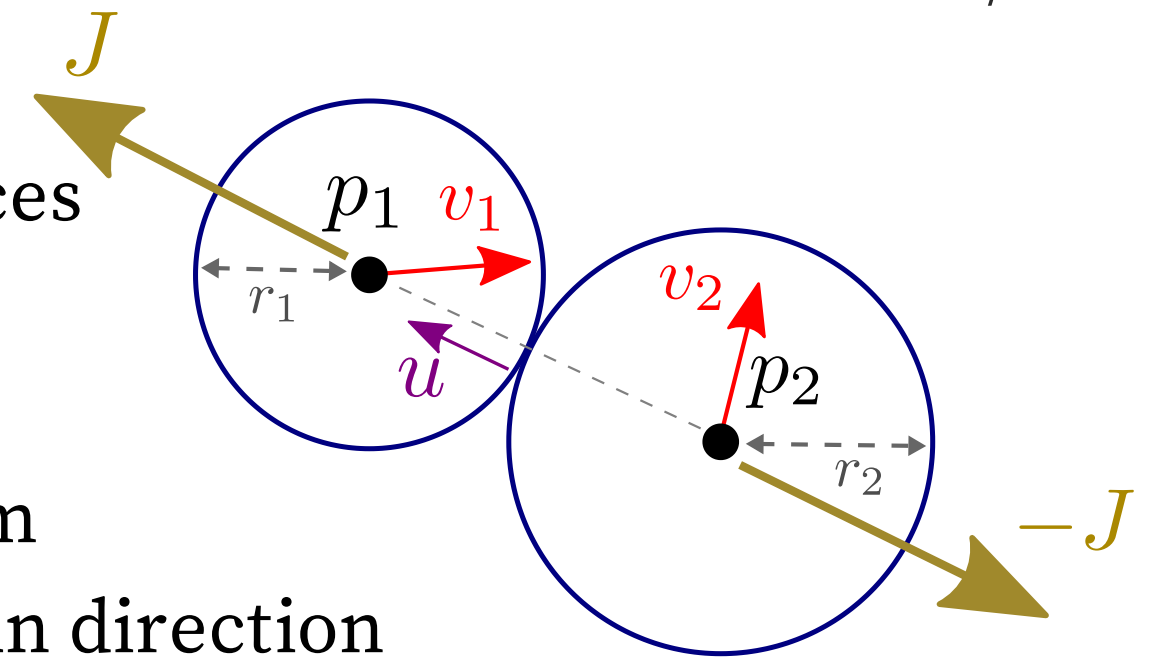
$$\Rightarrow j = 2 \frac{m_1 m_2}{m_1 + m_2} (v_2 - v_1) \cdot u$$

$$1/2 m_1 v_1^2 + 1/2 m_2 v_2^2 = 1/2 m_1 (v_1^{new})^2 + 1/2 m_2 (v_2^{new})^2$$

$$\Rightarrow m_1 v_1^2 + m_2 v_2^2 = m_1 \left(v_1 + \frac{j}{m_1} u \right)^2 + m_2 \left(v_2 - \frac{j}{m_2} u \right)^2$$

$$\Rightarrow 0 = 2 j v_1 \cdot u + \frac{j^2}{m_1} - 2 j v_2 \cdot u + \frac{j^2}{m_2}$$

$$\Rightarrow j = \frac{2}{1/m_1 + 1/m_2} (v_2 - v_1) \cdot u$$



Two spheres in collision

91/139

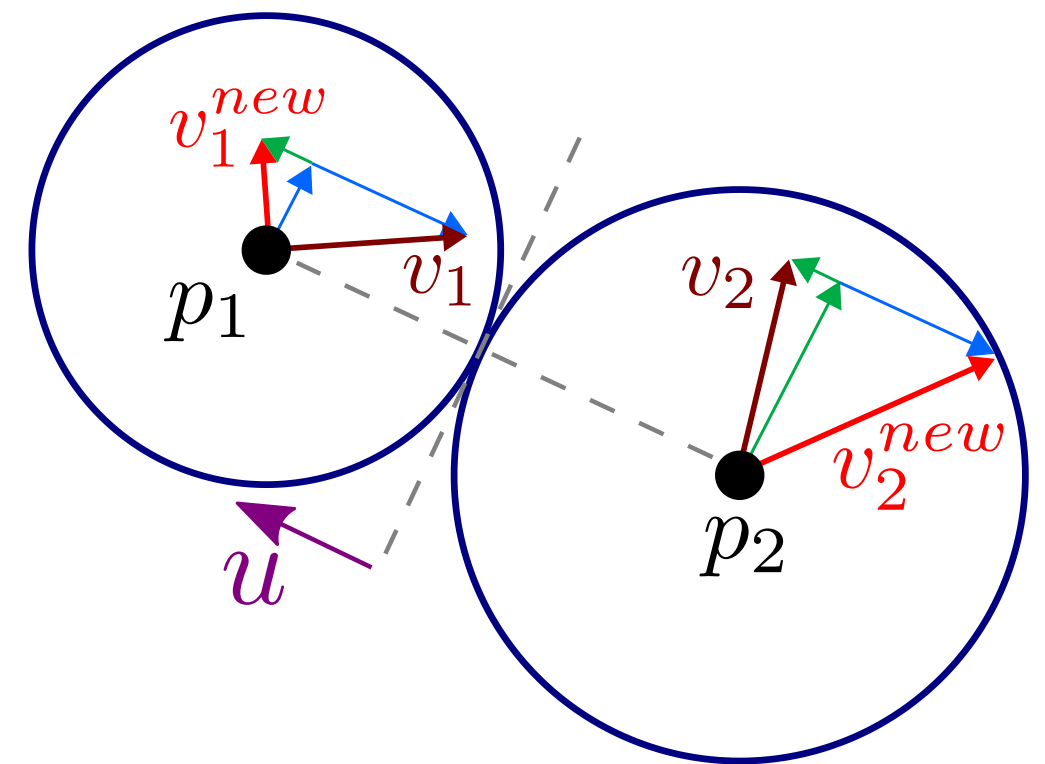
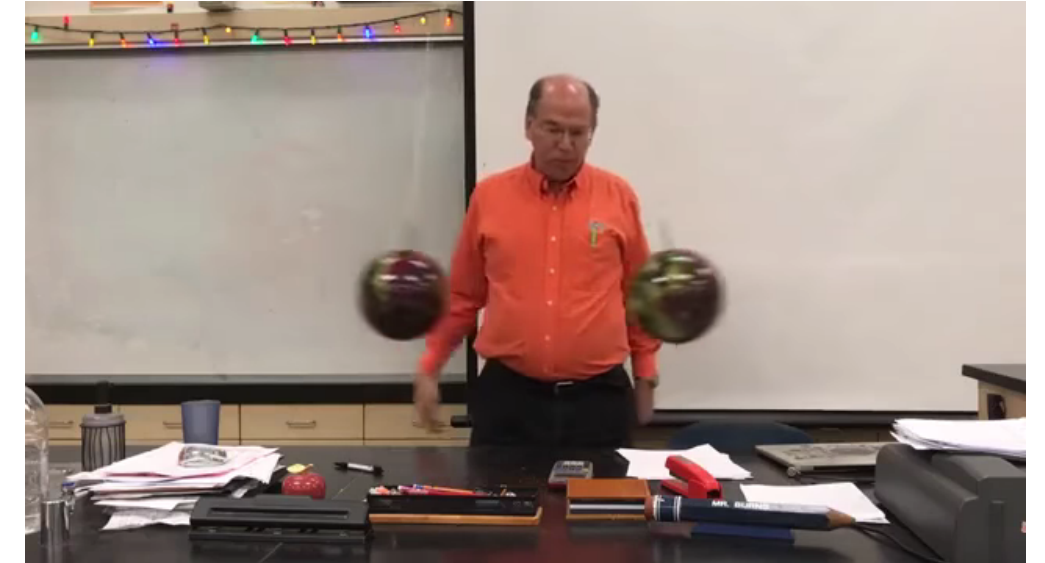
$$\begin{aligned} v_1^{new} &= v_1 + j/m_1 u = v_1 + 2 \frac{m_2}{m_1 + m_2} ((v_2 - v_1) \cdot u) u \\ v_2^{new} &= v_2 - j/m_2 u = v_2 - 2 \frac{m_1}{m_1 + m_2} ((v_2 - v_1) \cdot u) u \end{aligned}$$

Rem. If $m_1 = m_2$: Switch their $\perp$ speeds

$$\begin{aligned} v_1^{new} &= v_1 + ((v_2 - v_1) \cdot u) u = v_{1//} + v_{2\perp} \\ v_2^{new} &= v_2 - ((v_2 - v_1) \cdot u) u = v_{2//} + v_{1\perp} \end{aligned}$$

Can use restitution coefficient and attenuation $(\alpha, \beta) \in [0, 1]$

$$\begin{aligned} v_1^{new} &= \alpha v_{1//} + \beta v_{2\perp} \\ v_2^{new} &= \alpha v_{2//} + \beta v_{1\perp} \end{aligned}$$



Summary

92/139

1. Detect collision $\|p_1 - p_2\| \leq r_1 + r_2$

2-a. If collision (relative speed $> \epsilon$)

Elastic collision (/bouncing) $v_{1/2} = \alpha v_{1/2} \pm \beta J / m_{1/2}$

2-b. If *static* contact (relative speed $\leq \epsilon$)

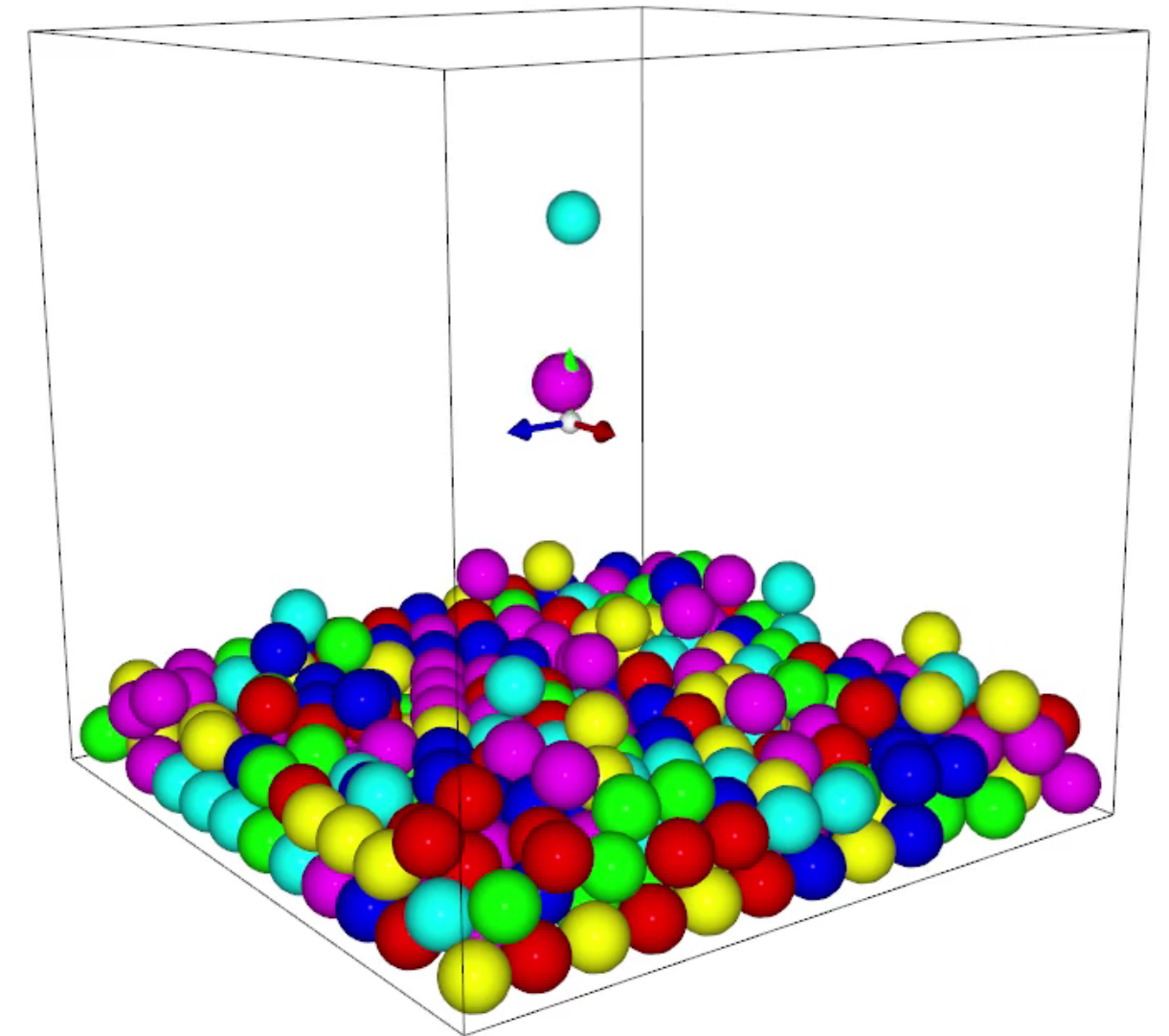
Friction $v_{1/2} = \mu v_{1/2}$, $\mu \in [0, 1]$

Avoids jittering

3. Correct position (project on contact surface)

$$p = p \pm d/2 u$$

$d = r_1 + r_2 - \|p_1 - p_2\|$: Collision depth



Multiple collisions

39

Pairwise collisions $\Rightarrow$ no global collision free state

- Correcting one collision may induce new collisions.
- Order of correction does matter

Reducing time-step help, Iterating over multiple pass help

But correct solution in all cases is complex $\rightarrow$ global approach

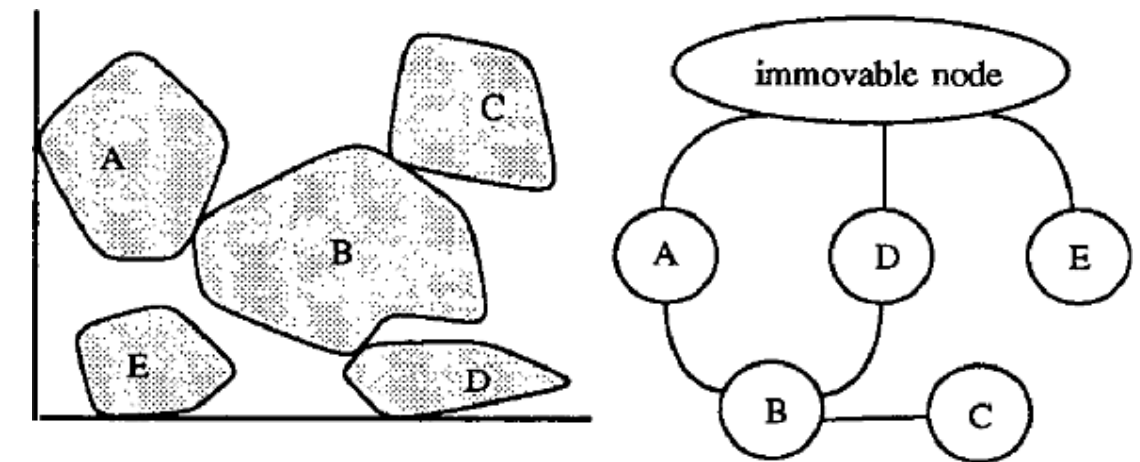
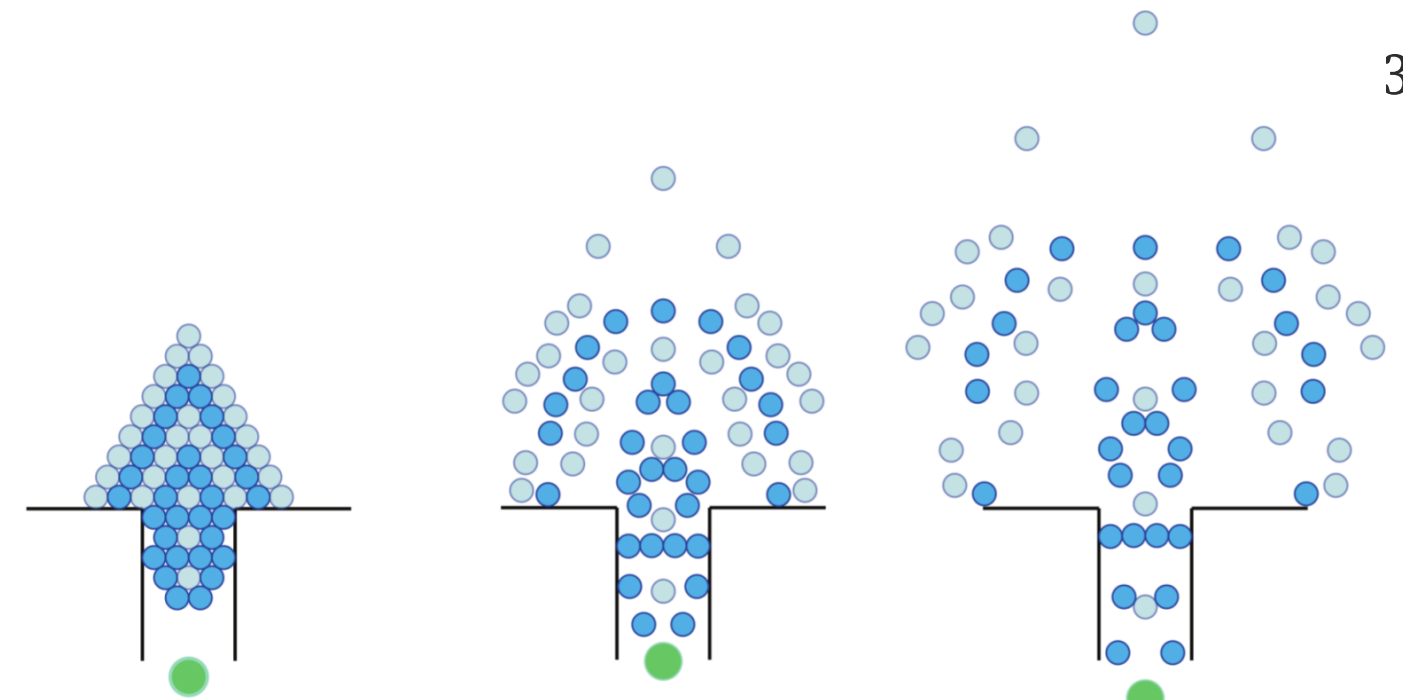
- Precompute contact graph
explicit shock propagation management
- Global constraint-based method

Impulse: $n_i \cdot (v_i - v_j) \geq 0$

Momentum preservation: $m_i v_i - m_j v_j = 0$

Energy preservation/dissipation

$\Rightarrow$ Linear Complementarity Program, Gauss Seidel, etc.



[*Realistic Animation of Rigid Bodies*. J. Hahn. SIGGRAPH 1988.]

[*Collision Detection and Response for Computer Animation*. M. Moore and J. Wilhelms. Computer Graphics 1988.]

[*Reflections on Simultaneous Impact*. B. Smith et al. SIGGRAPH 2012]

[*Guaranteed Resolution of Simultaneous Rigid Body Impact*. E. Vouga. ACM SIGGRAPH 2017]

Elastic models

Spring structure

Numerical solution of ODE

Cloth simulation

Material model

95/139

Elasticity: Shape goes back toward its original rest position when external forces are removed.

- Purely elastic models don't lose energy when deformed (potential $\leftrightarrow$ kinetic)



Plasticity: Opposite of elasticity. Plastic material doesn't come back to their original shape (/change their rest position during deformation).

- Ductile material - can allow large amount of plastic deformation without breaking (plastic)
- Brittle - Opposite (glass, ceramics)

Viscosity: Resistance to flow (usually for fluid, ex. honey)

In reality

- *Elasto-plastic materials:* Allow elastic behavior for small deformation, and plastic at larger one.
- *Visco-elastic materials:* Elastic properties with delay.

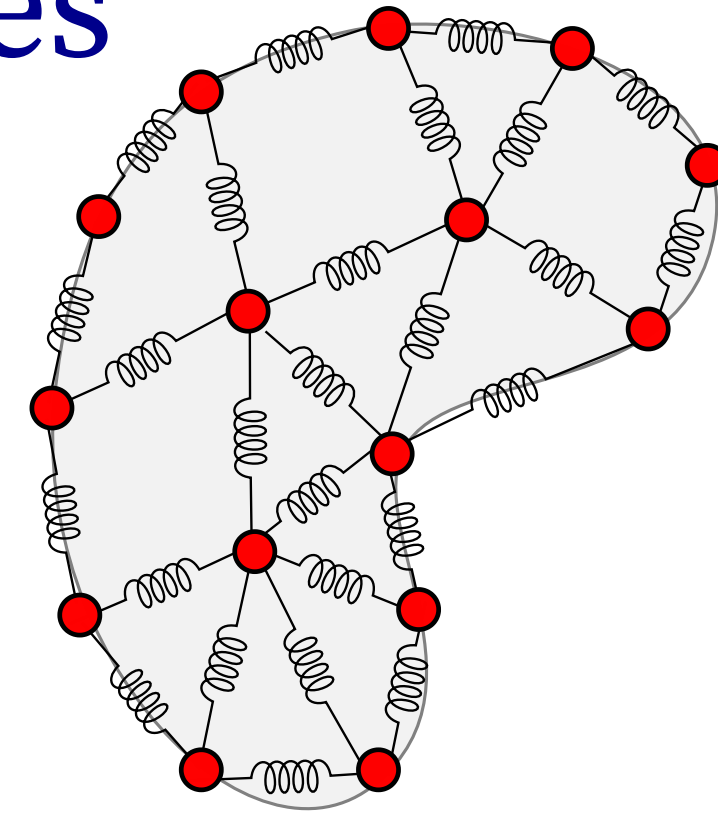


Modeling elastic shapes with particles

96/139

Spring mass systems

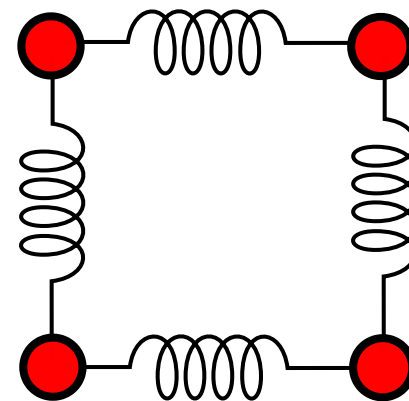
- Particles (position, velocity, mass): samples on shape
- Springs : link closed-by particles in the reference shape



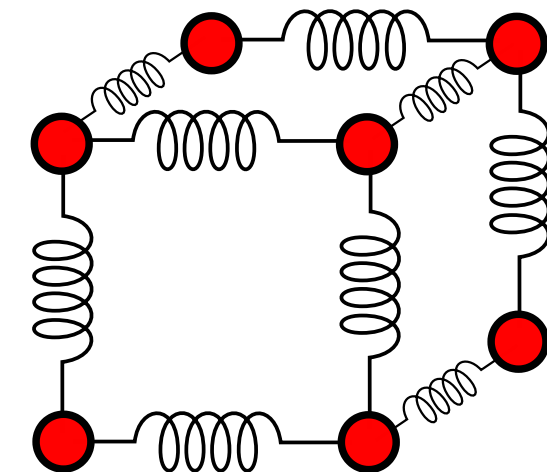
1D curve structure



2D surface structure



3D volume structure



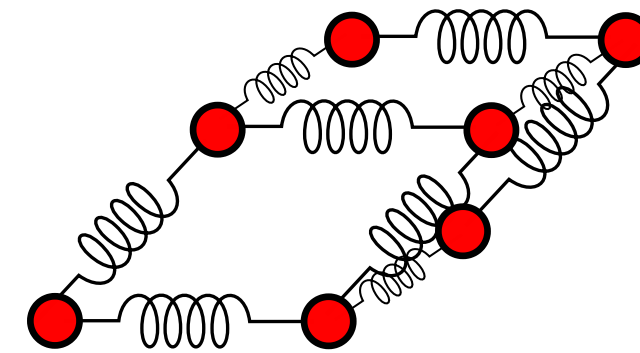
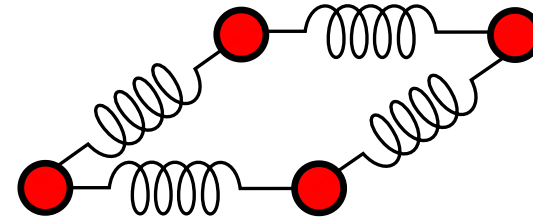
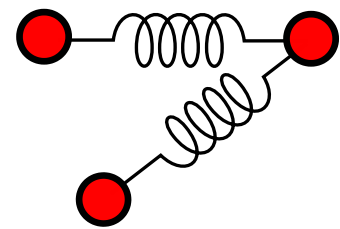
Spring structure

97/139

How to model spring connectivity ?

- **Structural springs:** 1-ring neighbors springs ($\simeq$ mesh edges)

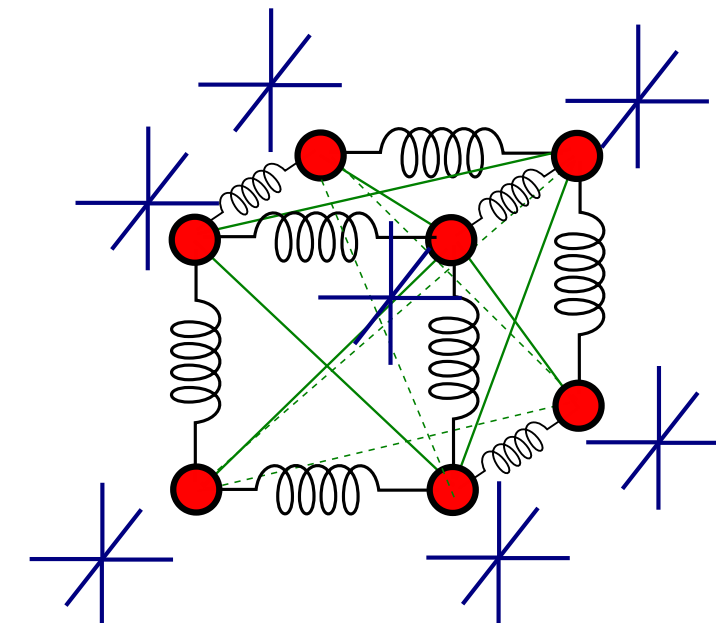
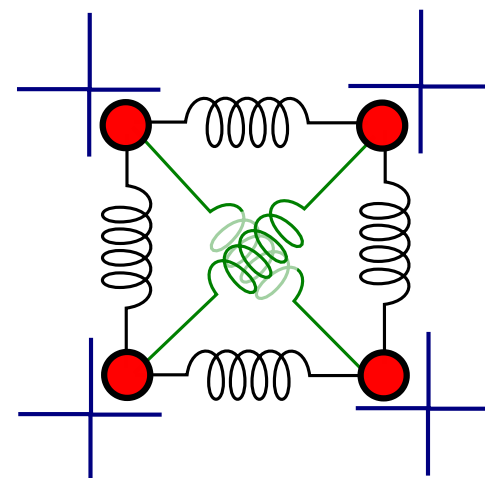
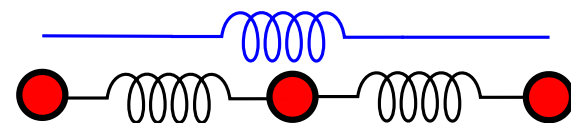
(+) Limit elongation/contraction, (-) Allows shearing, and bending



$\Rightarrow$ Add extra springs connectivity

- **Shearing springs:** Diagonal links

- **Bending springs:** 2-ring neighborhood

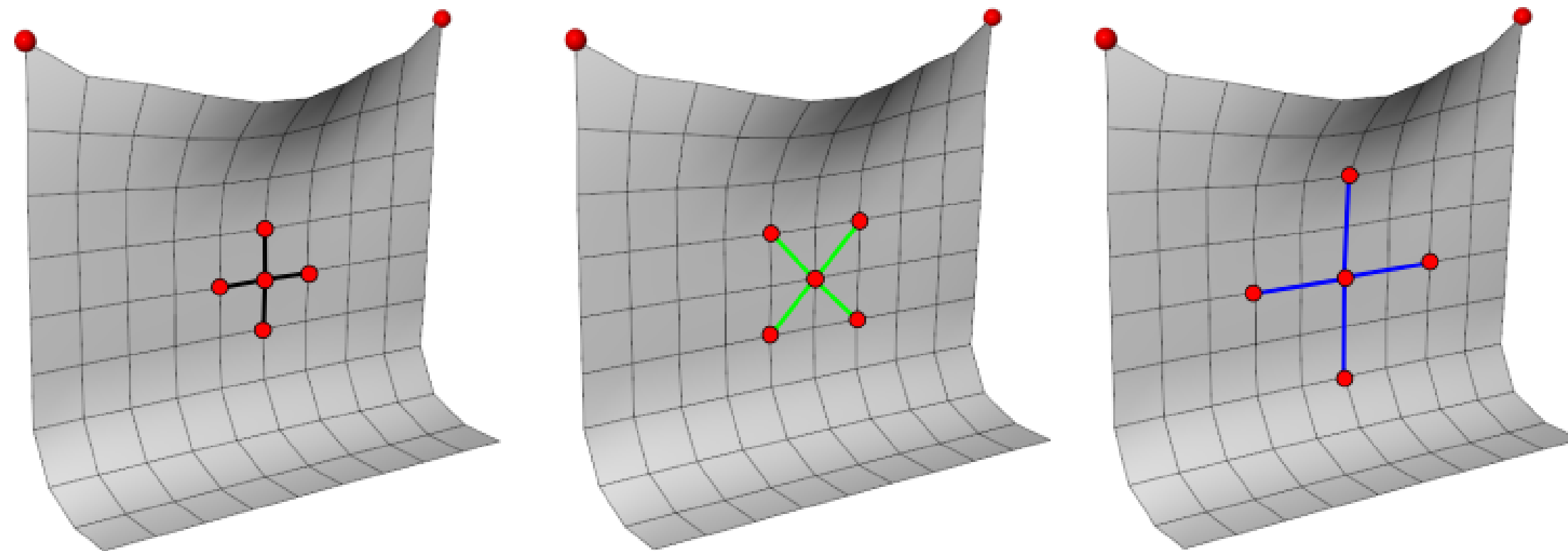


Cloth Simulation

Mass-spring cloth simulation

99/139

- Particles are sampled on a $N \times N$ grid.
 - Each particle has a mass m ($m_{cloth} = N^2 m$)
- Set structural, shearing and bending springs.



Forces

100/139

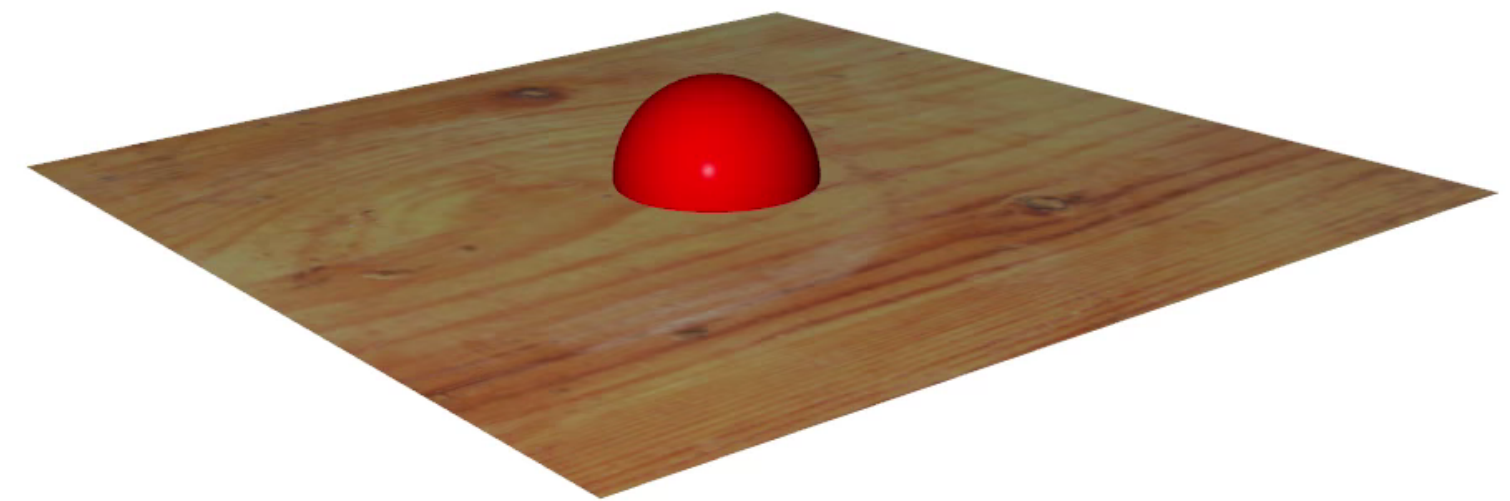
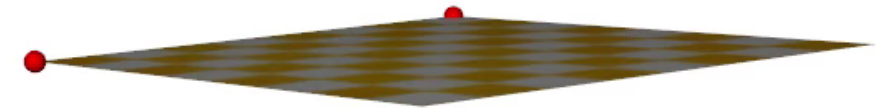
- On each particle: gravity + drag + spring forces

$$- F_i(p, v, t) = m_i g - \mu v_i(t) + \sum_{j \in \mathcal{V}_i} K_{ij} (\|p_j(t) - p_i(t)\| - L_{ij}^0) \frac{p_j(t) - p_i(t)}{\|p_j(t) - p_i(t)\|}$$

- $\mathcal{V}_i$: neighborhood of particle i

- L_{ij}^0 : rest length of spring ij

Associated ODE $\forall i, \begin{cases} p_i'(t) = v_i(t) \\ v_i'(t) = F_i(p, v, t)/m_i \end{cases}$

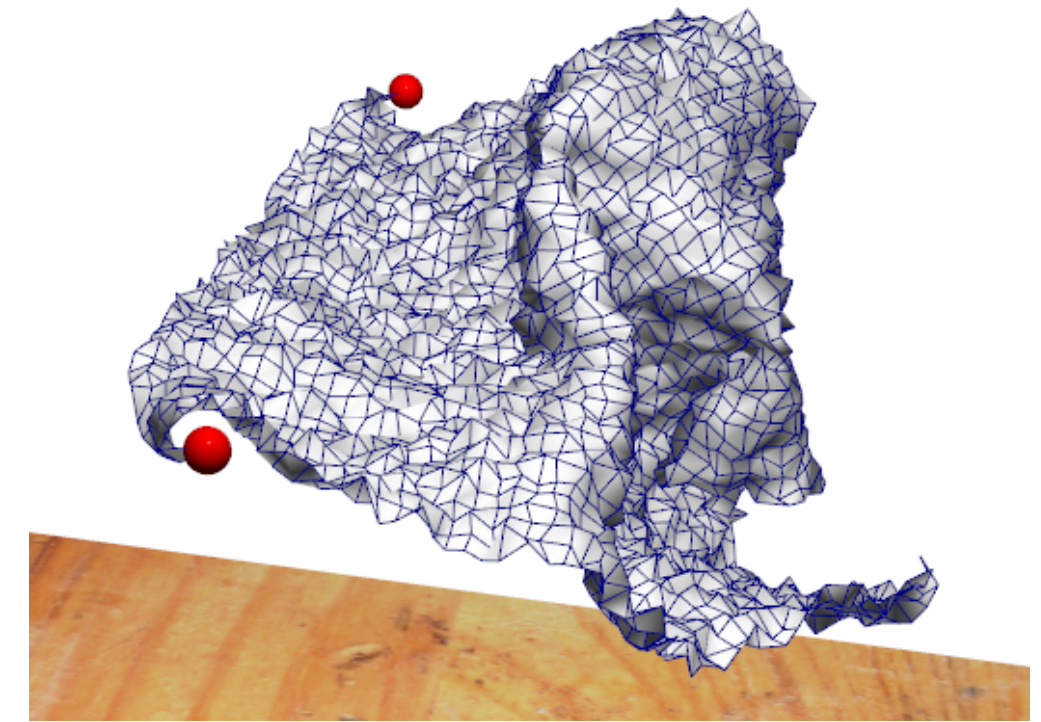


Q. How can we model the effect of the wind ?

Note on Mass-Spring numerical solution

101/139

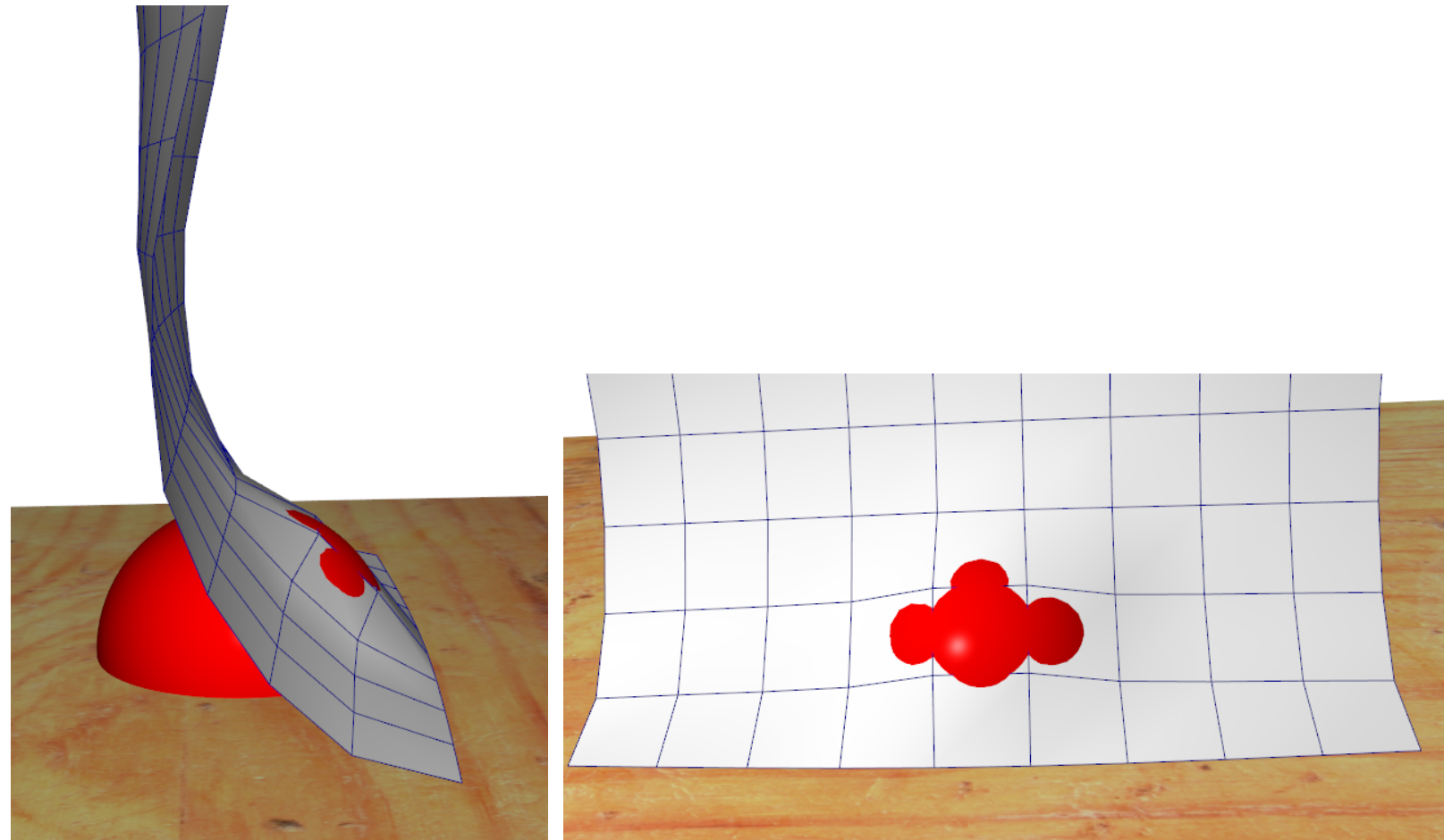
- Non-linear ODE
- Large K_{ij} : good length preservation, but stiff ODE
⇒ divergence of explicit schemes.
- Avoid explicit Euler (divergence)
- Semi-implicit Euler/Verlet works fine for low K_{ij}
Semi-implicit Euler + PBD allows simple integration + stable stiff springs
[Muller et al. [PBD](#) , [Inextensible clothing in Computer Games](#)]
- RK4 more accurate (but higher complexity than Verlet)
- Implicit Euler : requires linearization, but very stable



Collisions

102/139

- Simple approach : Handled as collision between particles and shapes
 - (+) Simple and efficient
 - (-) Collision may still appears within a triangle
 - ⇒ Exhaustive approach: edges + faces



Limitation of mass spring model and continuous model

103/139

- Does mass-spring system converge toward a unique solution when sampling increase ?

⇒ No :(

Depends on the connectivity → bad for physical accuracy

Corollary

- Mass-springs work well for grid-mesh structure (draping)
- Less for arbitrary triangular meshes

1st improvment: Change toward energy formulation for bending springs (limits locking effect)

$$F = \frac{\partial E}{\partial p}$$

$$E = \frac{1}{2} K L \kappa^2, \kappa: \text{curvature}$$

[Cho et al, Stable but Responsive Cloth, ACM SIGGRAPH 2002]



Triangle as continuous elements

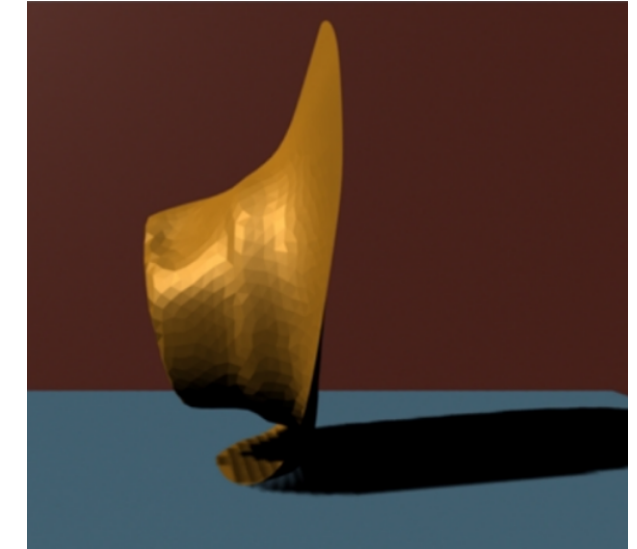
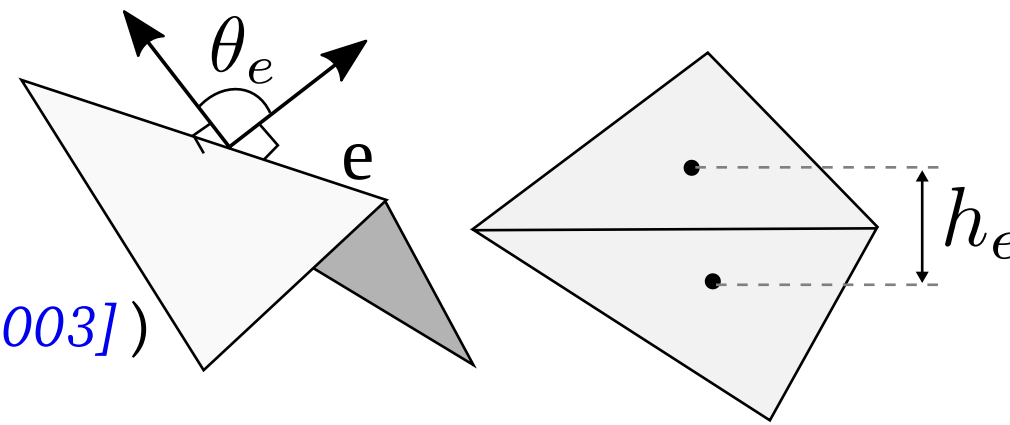
104/139

- Defining Bending Energy between triangles

$$- W_B(x) = \sum_{\text{edges } e} (\theta_e - \theta_e^0) \frac{\|e^0\|}{h_e^0}$$

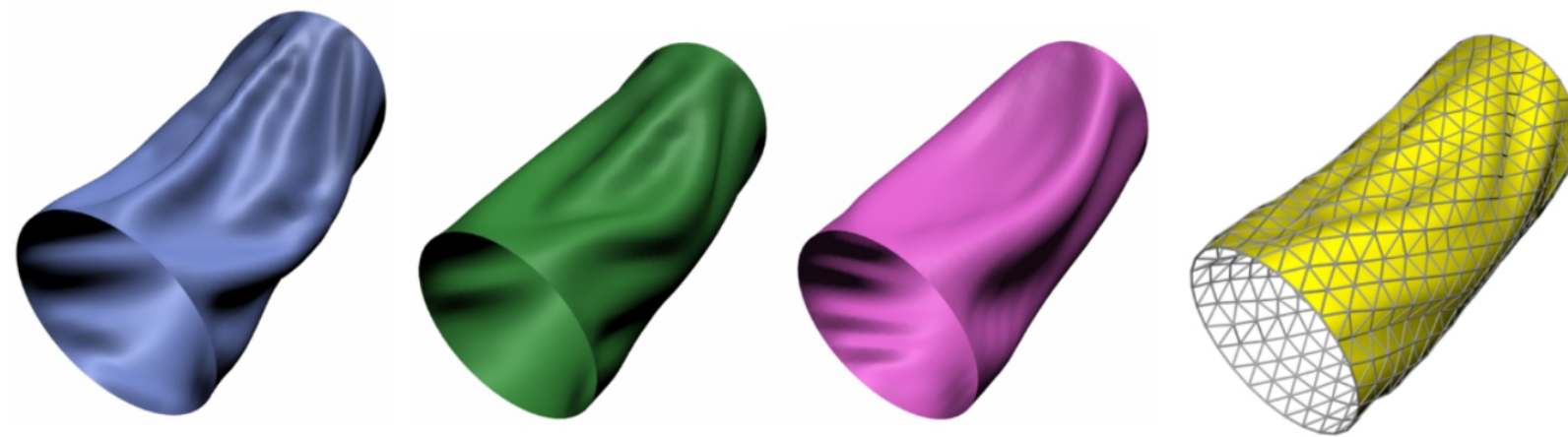
[E. Grinspun et al., Discrete Shells, SCA 2003]

(or expressed using forces in [R. Bridson et al., SCA 2003])



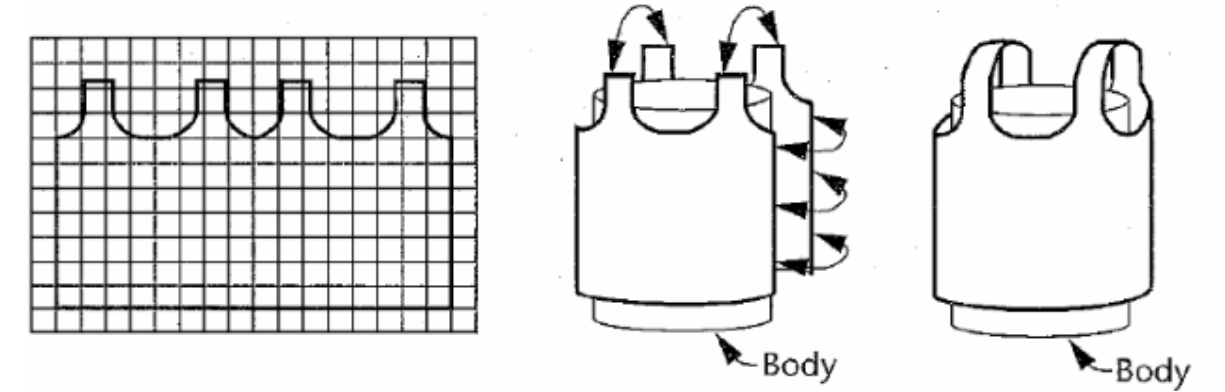
- Going toward full FEM numerical resolution

- B. Thomaszewski et al. [SCA 2006], [VRIPHYS 2008], [EG 2009].

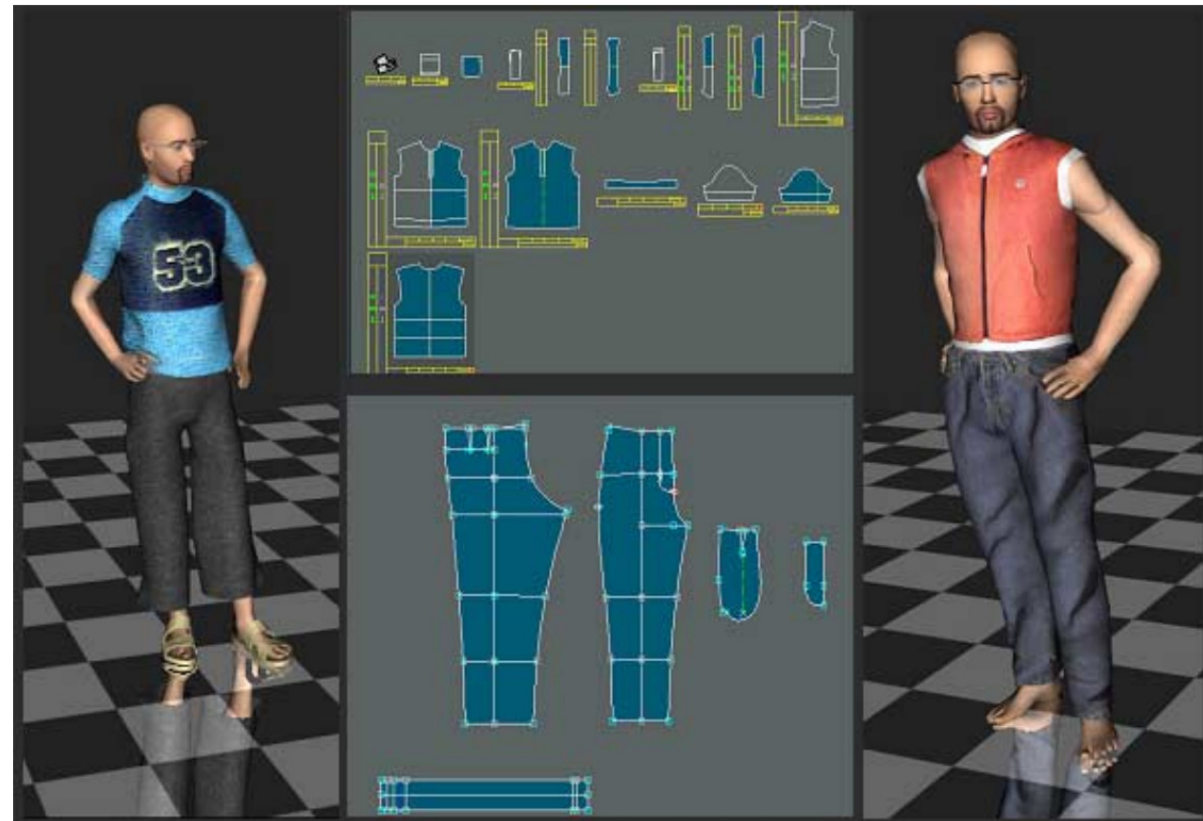


Clothing

- Stitch 2D patterns together to generate full cloth
- Cloths are developable material (preserve length w/r their 2D patterns)



39

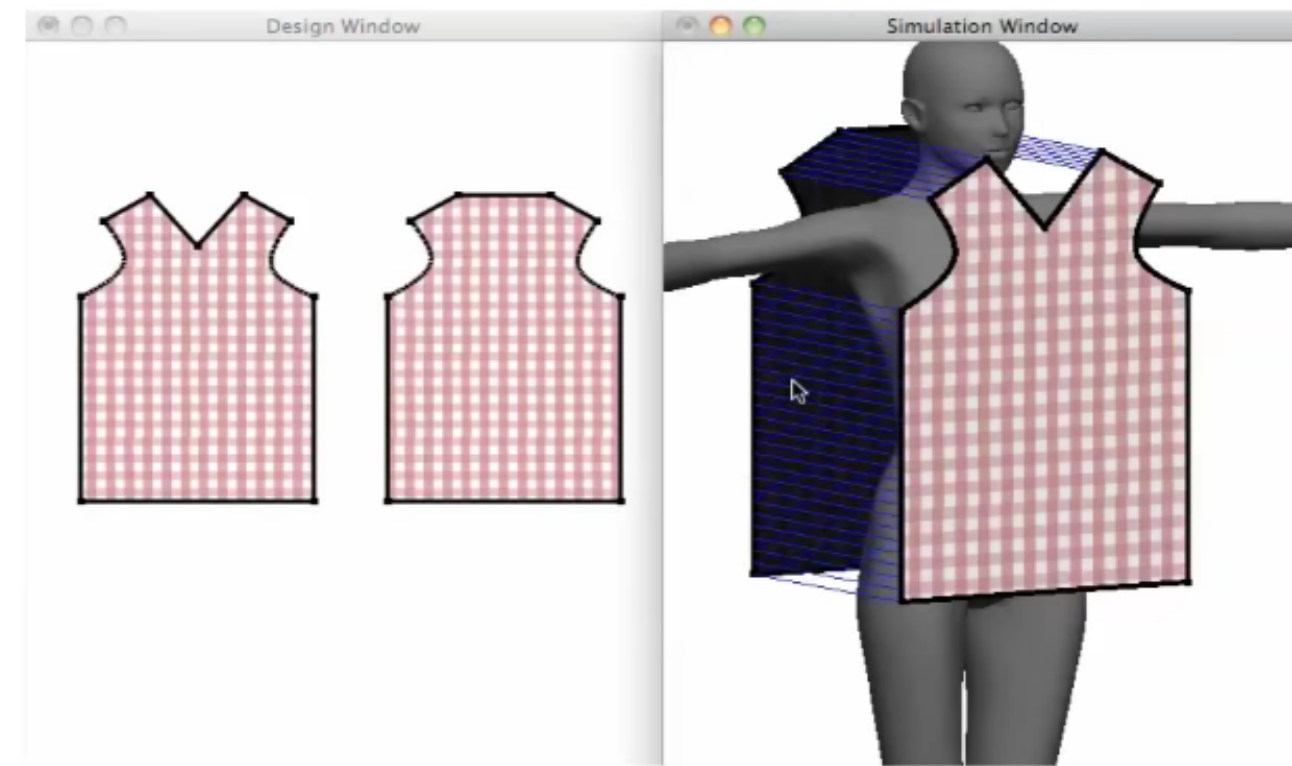


[Thalman et al. 2002]

Overview

Cloth Pattern

Cloth Simulation



[Umetani et al., 2011]

Detecting self collision

Handled as moving point in collision with moving triangle

Inputs

- Triangle $P_1(t)P_2(t)P_3(t)$, a point $P(t)$
- Each position $P_k(t) = P_k(0) + t v_{P_k}$

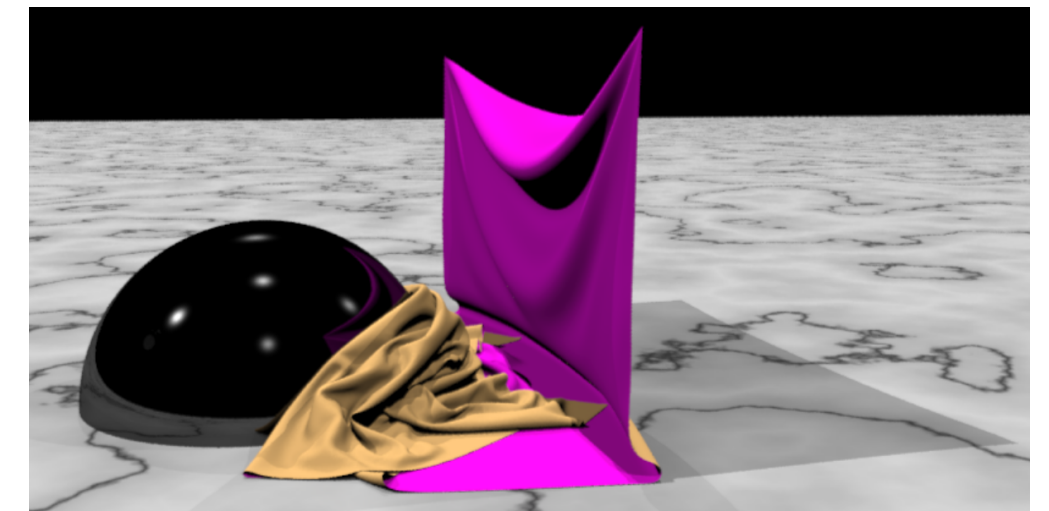
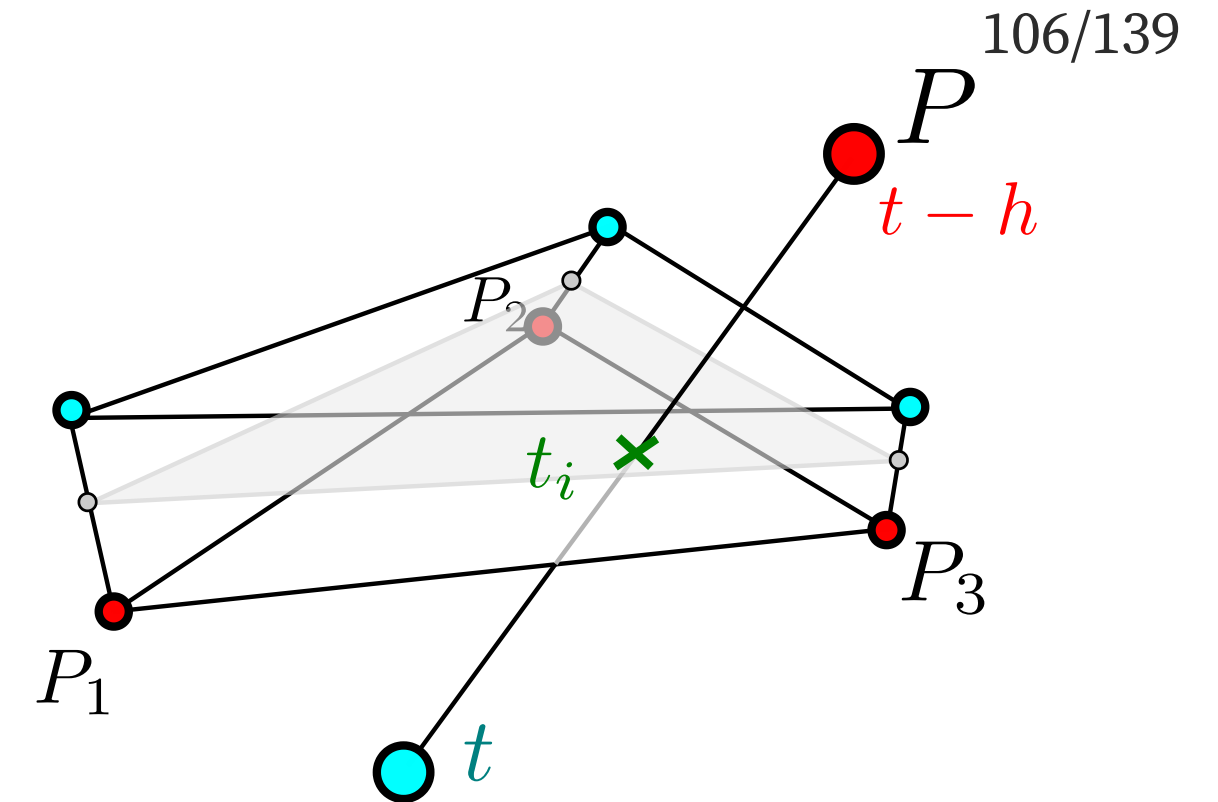
Computing intersection

Necessary condition

- Find $t_i \in [0, h]$ such that $P(t_i)$ is in triangle plane
 $(P(t_i) - P_1(t_i)) \times n(t_i) = 0$
 $n(t_i)$: normal of the triangle at time t_i

Sufficient condition

- Check $P(t_i)$ is inside the triangle
 $P(t_i) = \alpha P_1(t_i) + \beta P_2(t_i) + \gamma P_3(t_i)$
 $(\alpha, \beta, \gamma) \in [0, 1]^3, \alpha + \beta + \gamma = 1$



[X. Provot. Collision and self-collision handling in cloth model dedicated to design garments. Graphics Interface 1997.]

[R. Bridson et al. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. ACM SIGGRAPH 2002]

Animating Virtual Characters

Skin deformation

Skeleton based deformation - Skinning

109/139

Objective: Deform articulated character

Idea: Use skeleton to control limbs

Articulations as rotations

Animation Skeleton

Set of frames T_i : position, orientation

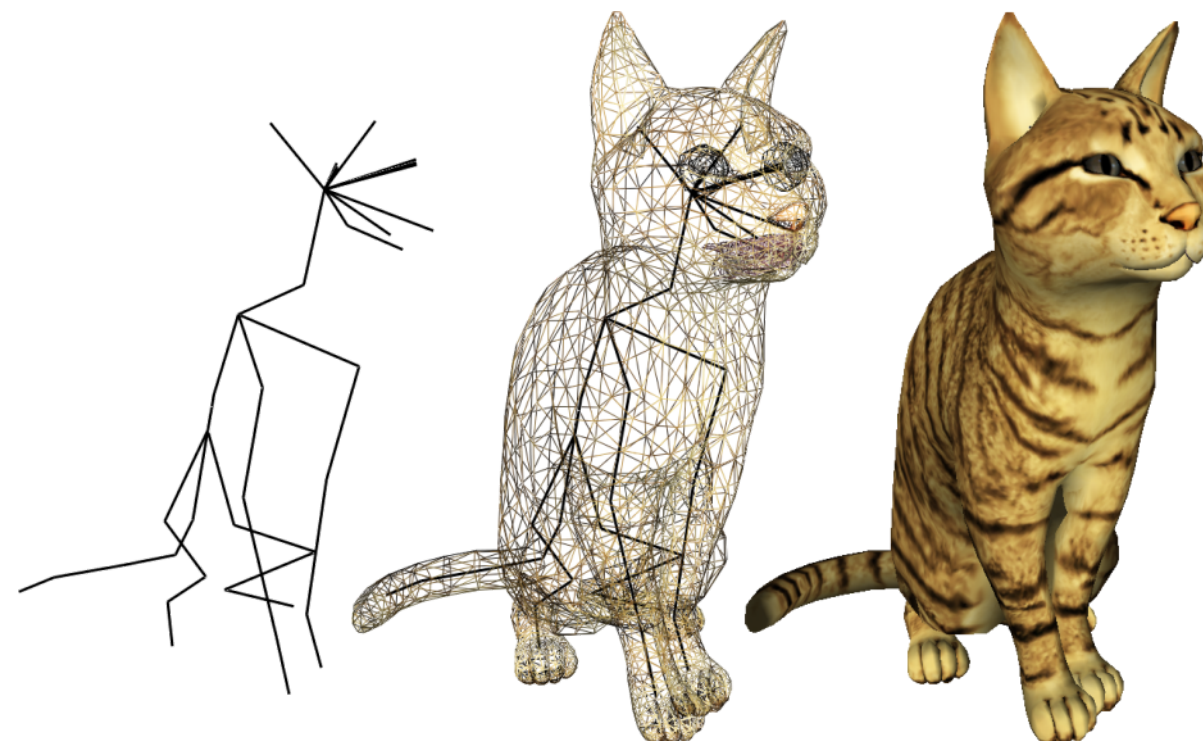
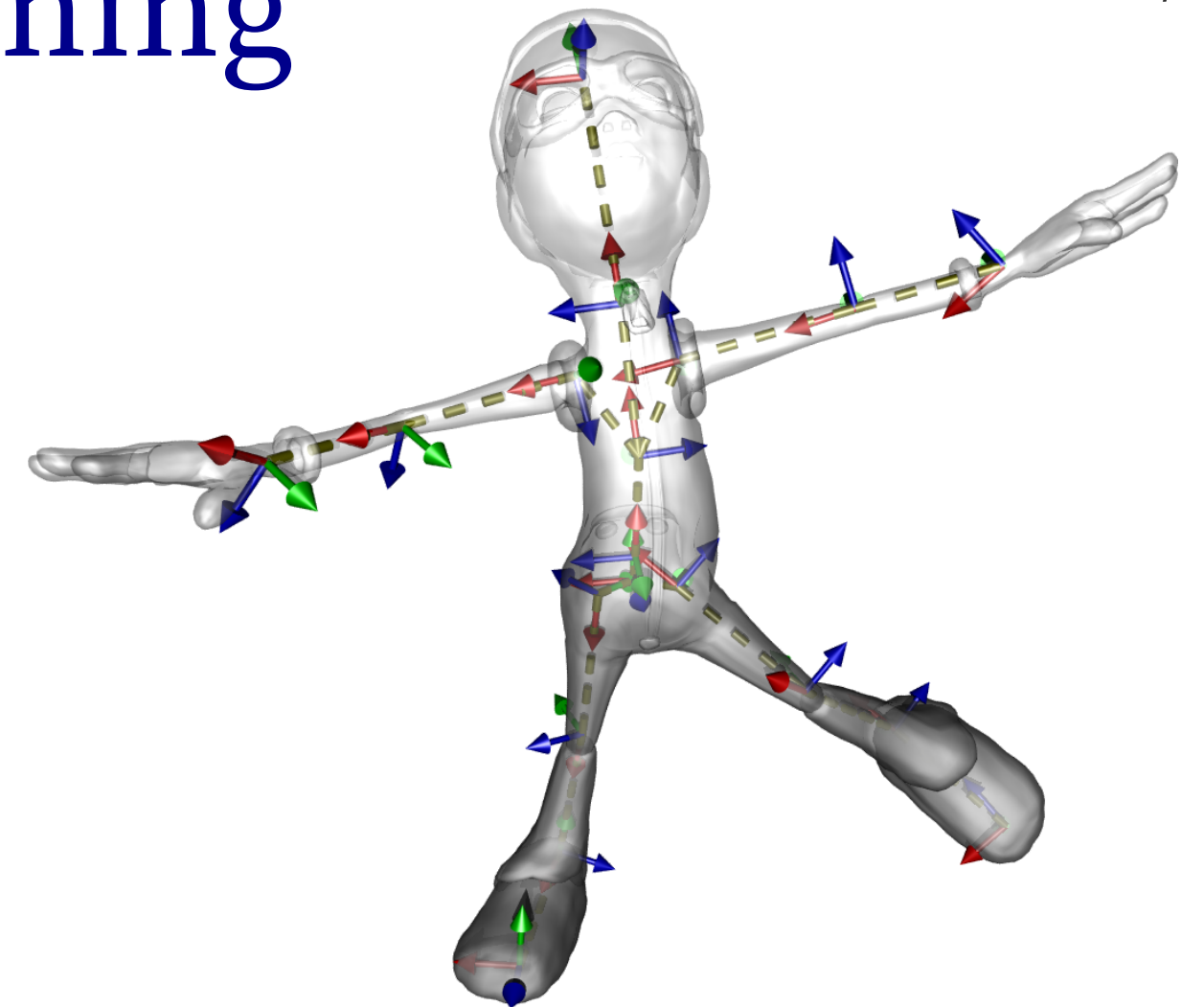
Describe non-rigid parts of the character (dof)

Terminology

Joint = A frame T_i

Bone = Segment between two joints

Note: Animation skeleton $\neq$ Anatomical skeleton



Rigid Skinning

110/139

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

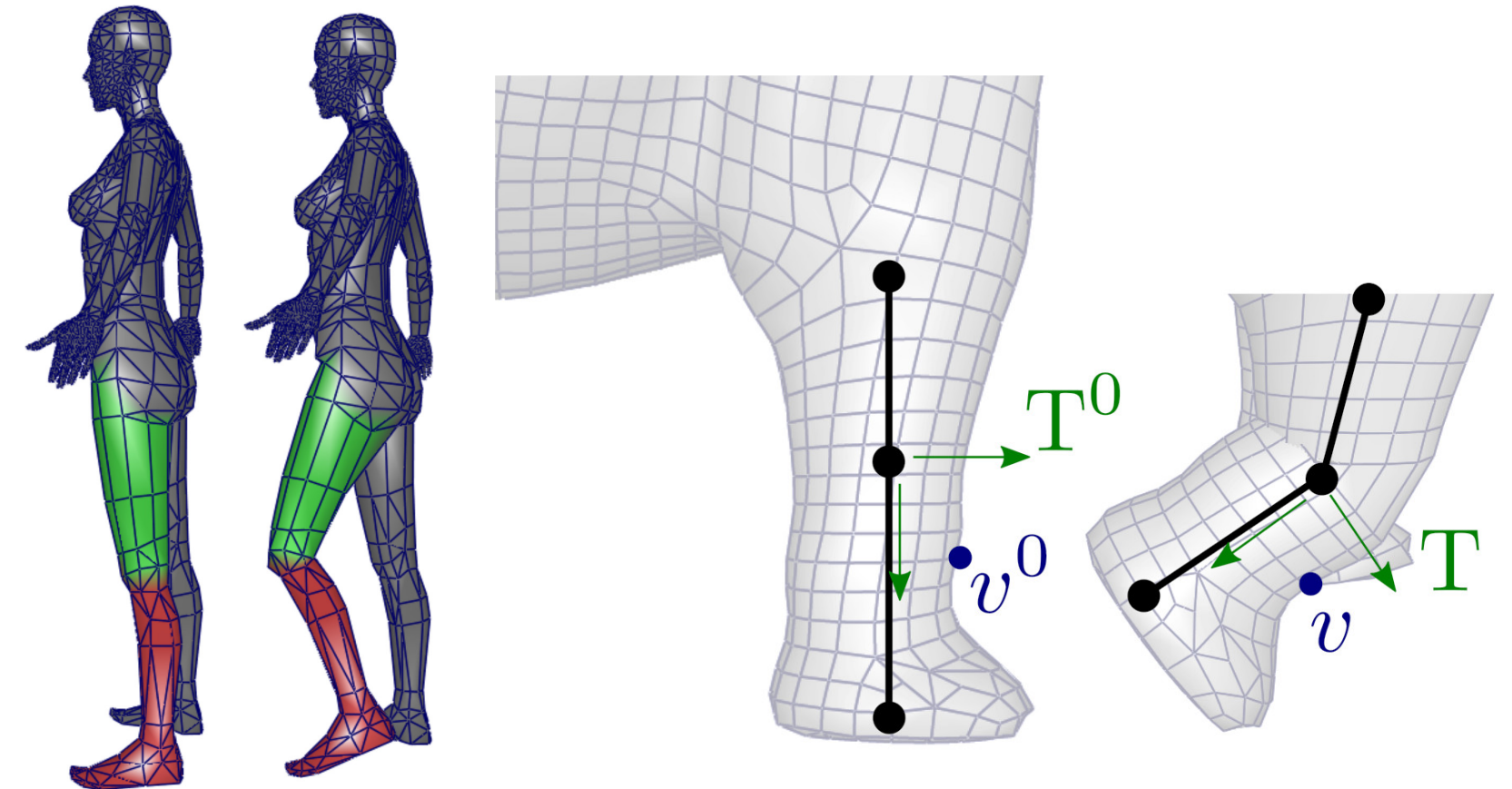
Deformation formulation

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

111/139

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

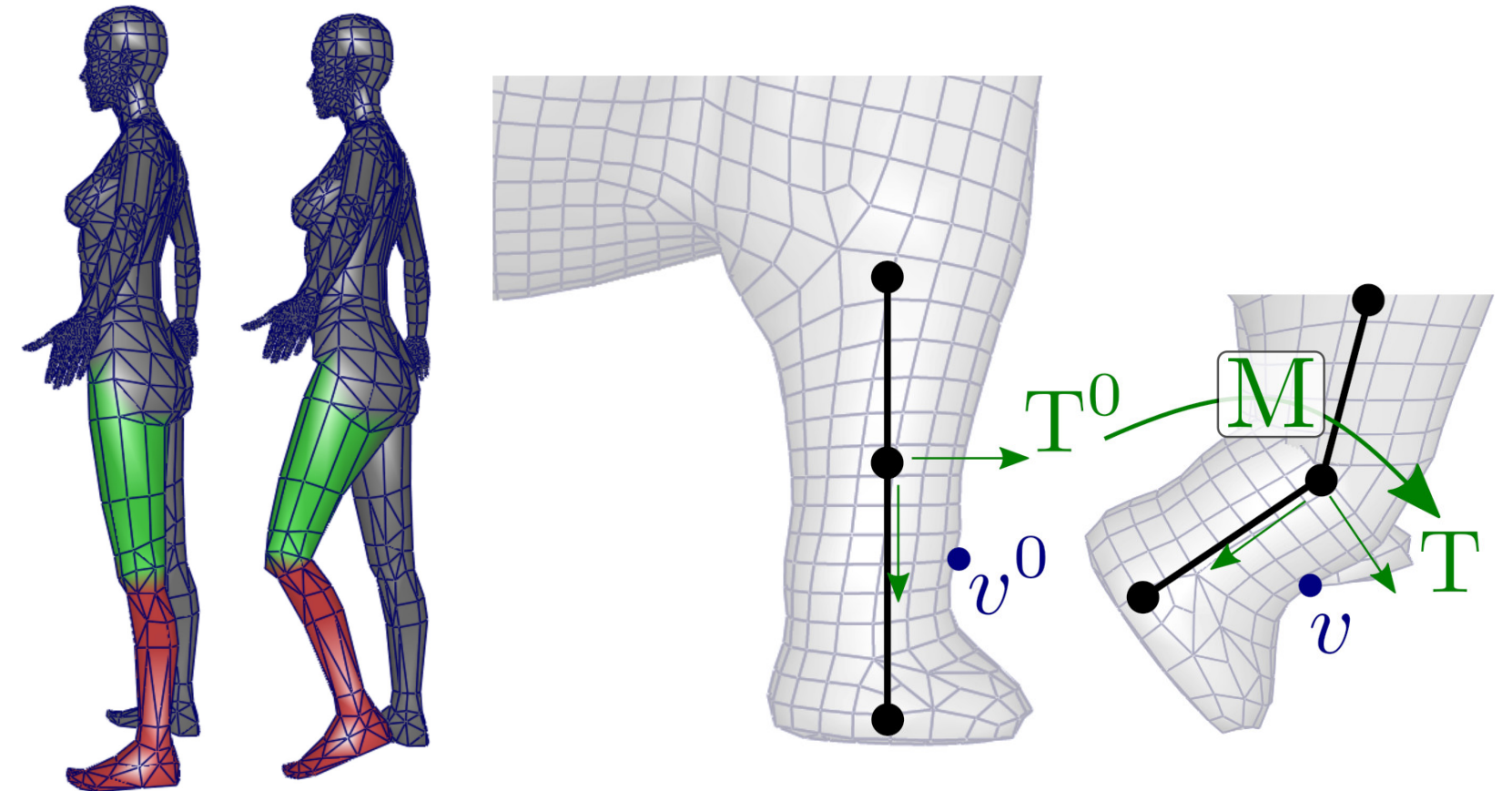
Deformation formulation

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

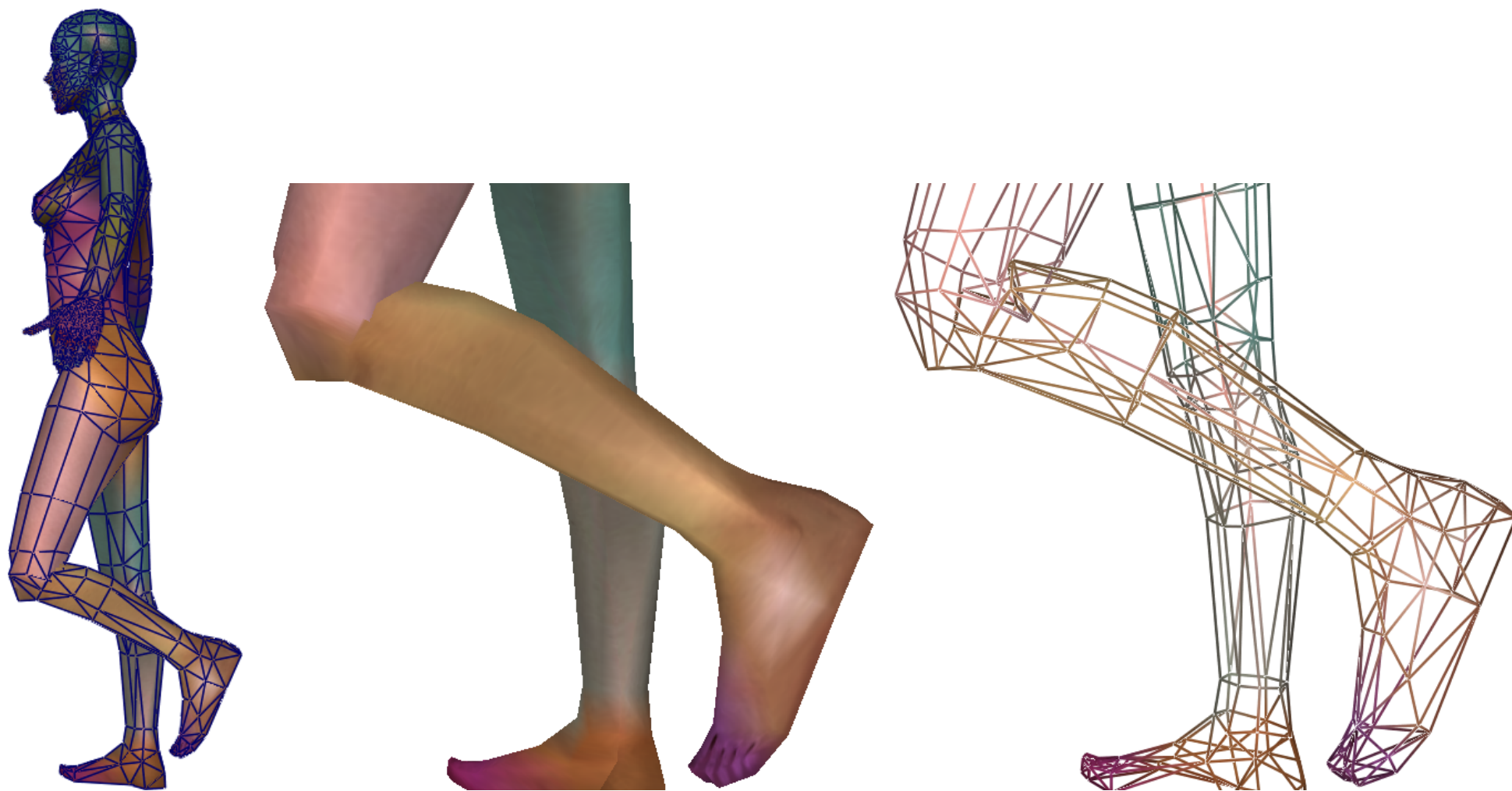
$$\Rightarrow T^{-1}v = (T^0)^{-1}v_0$$

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

Rigid Skinning

112/139

- (+) 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

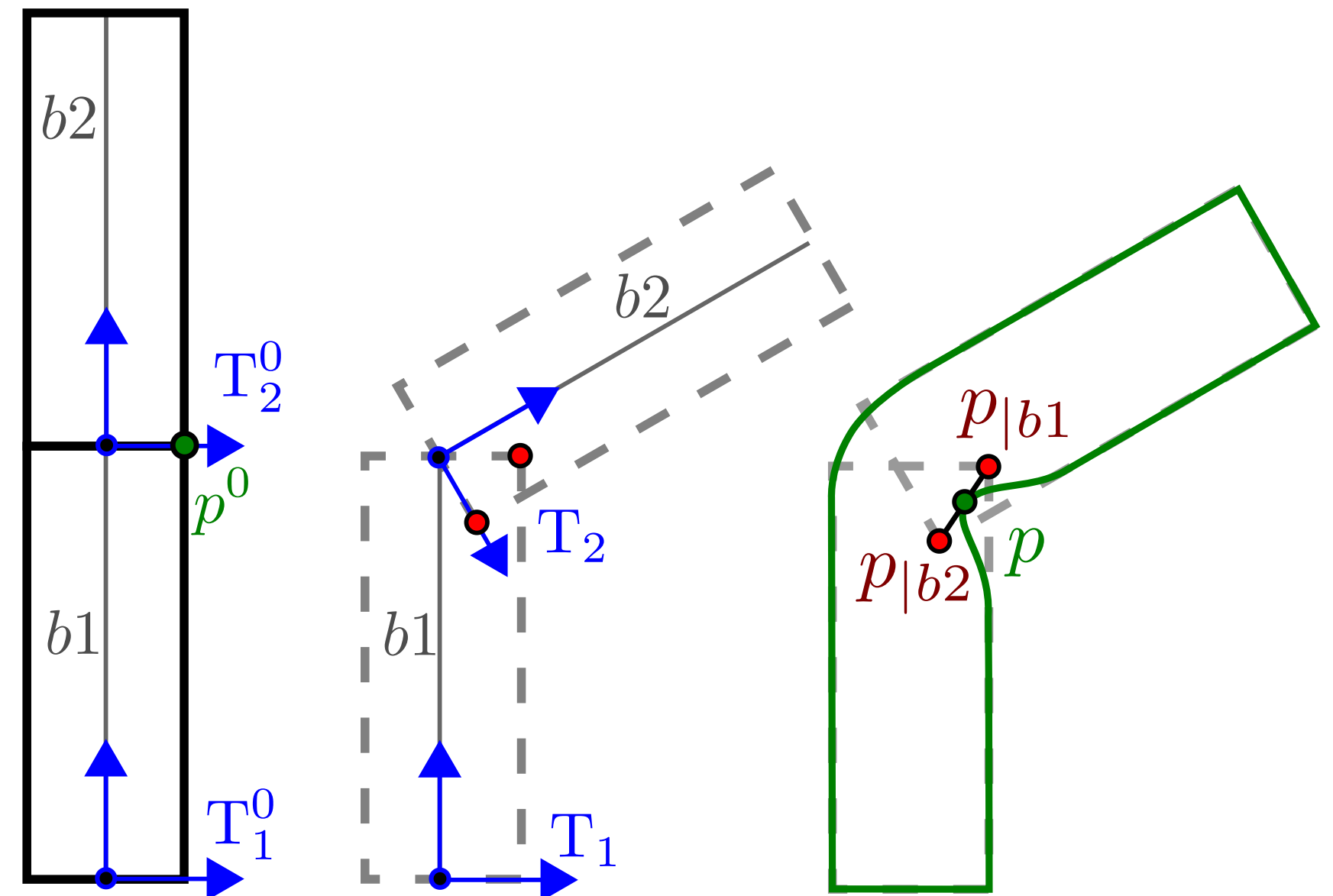
Smooth skinning

113/139

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

Example at middle vertex position of a bending cylinder

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



Smooth skinning

114/139

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

Example at middle vertex position of a bending cylinder

$$p = 0.5 p_{|b1} + 0.5 p_{|b2}$$

$$p = 0.5 \underbrace{T_1 (T_1^0)^{-1}}_{M_1} p^0 + 0.5 \underbrace{T_2 (T_2^0)^{-1}}_{M_2} p^0$$

$$p = (0.5 M_1 + 0.5 M_2) p^0$$

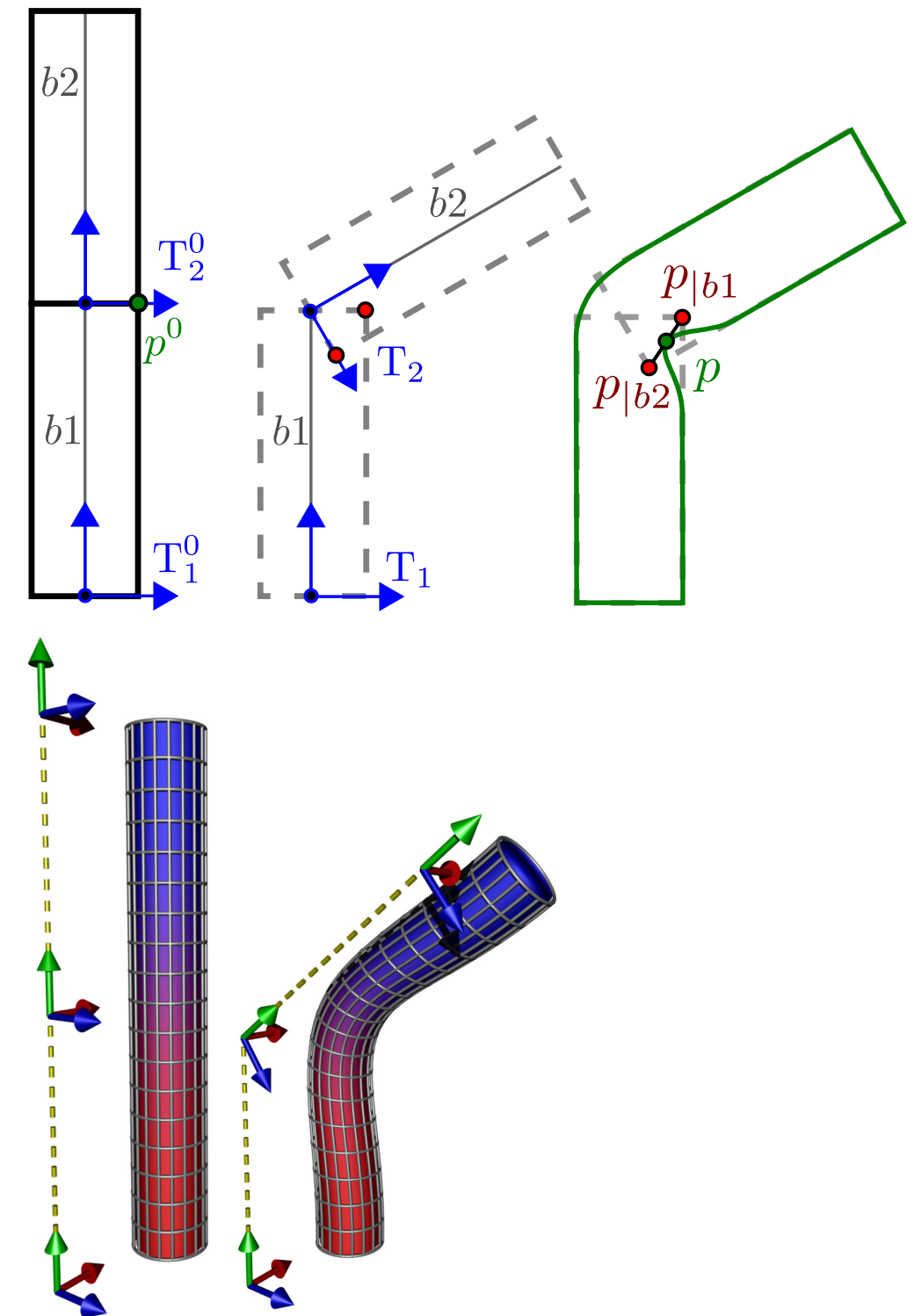
Can be generalized to arbitrary interpolation between two bones

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

α : skinning weights

Can be generalized to any number of bones

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

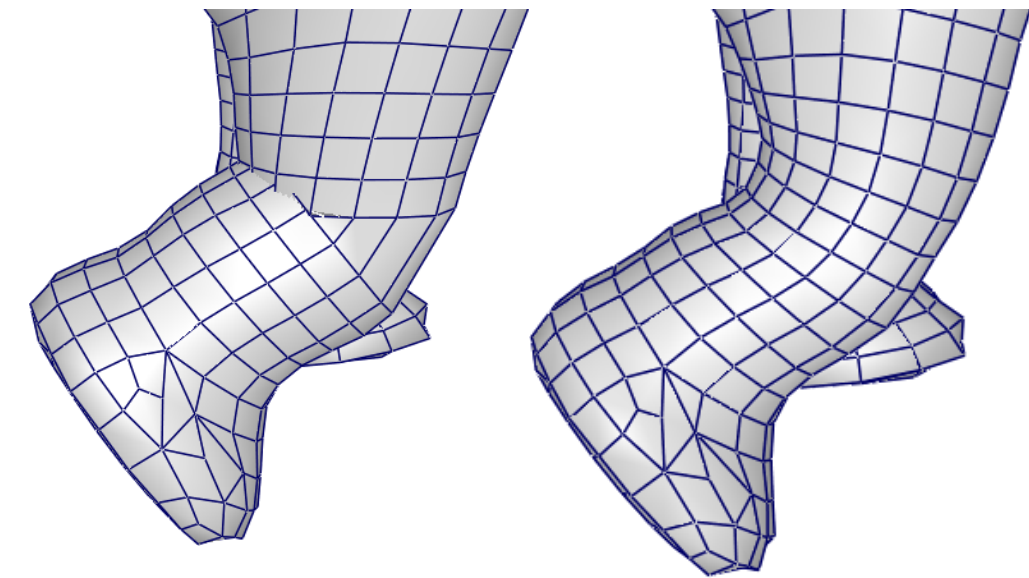
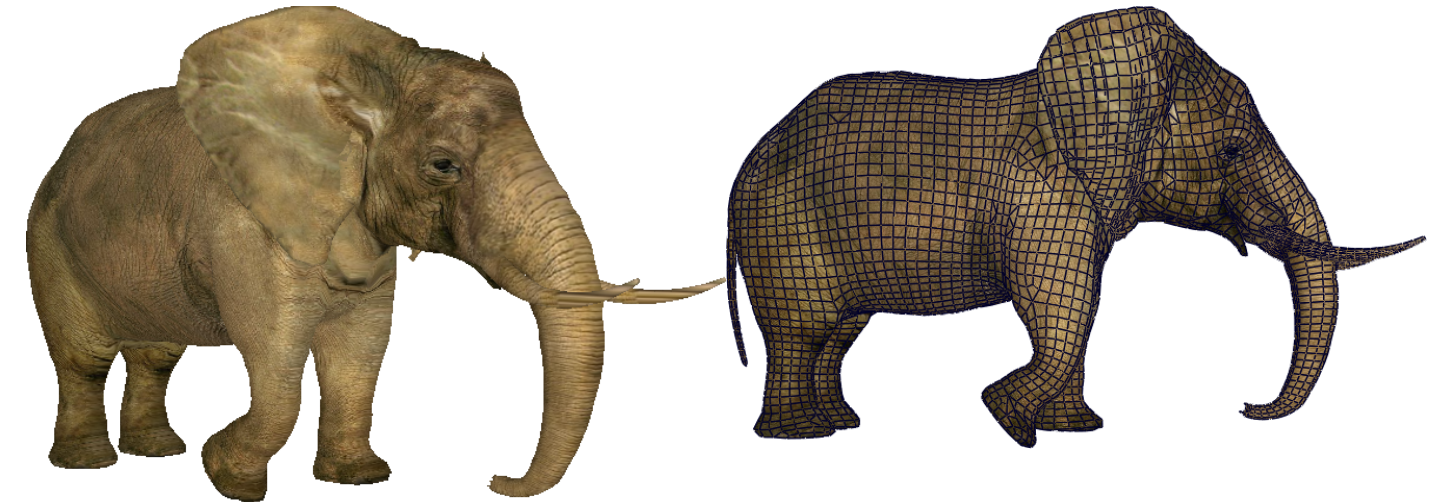


Smooth skinning - Summary

115/139

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

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



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 Laperrière, Daniel Thalmann. *Graphics Interface*, 1988]

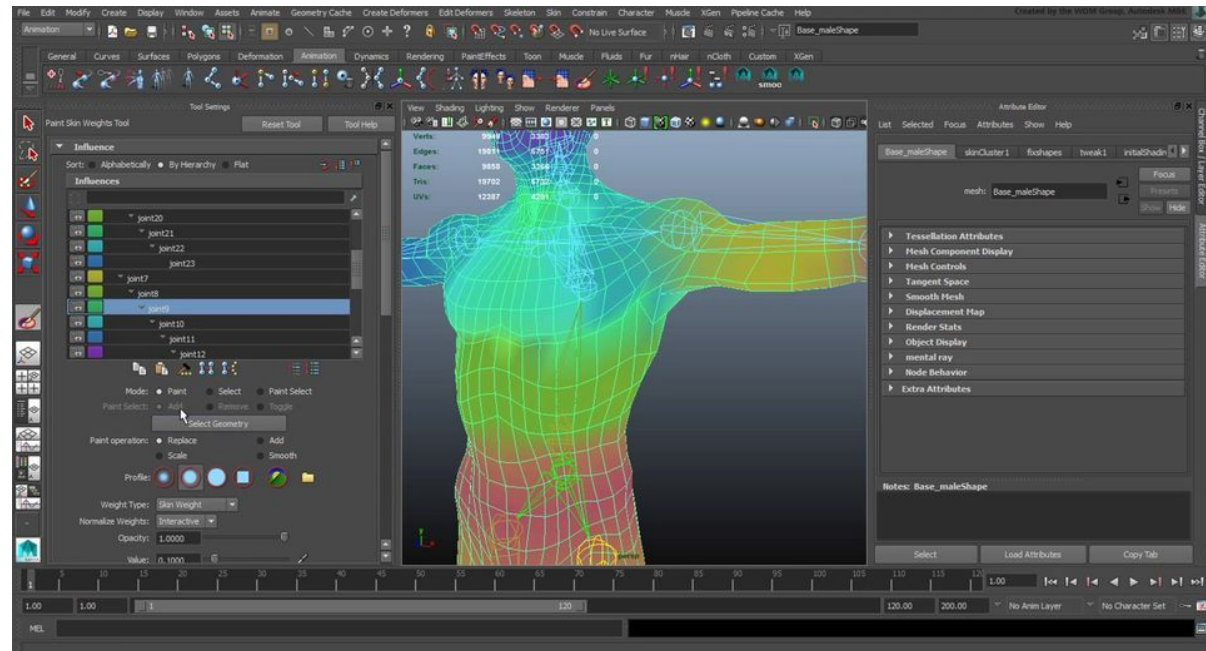
[*Over My Dead, Polygonal Body*. Jeff Lander. *Game Developer Magazine*, 1998]

Skinning weights

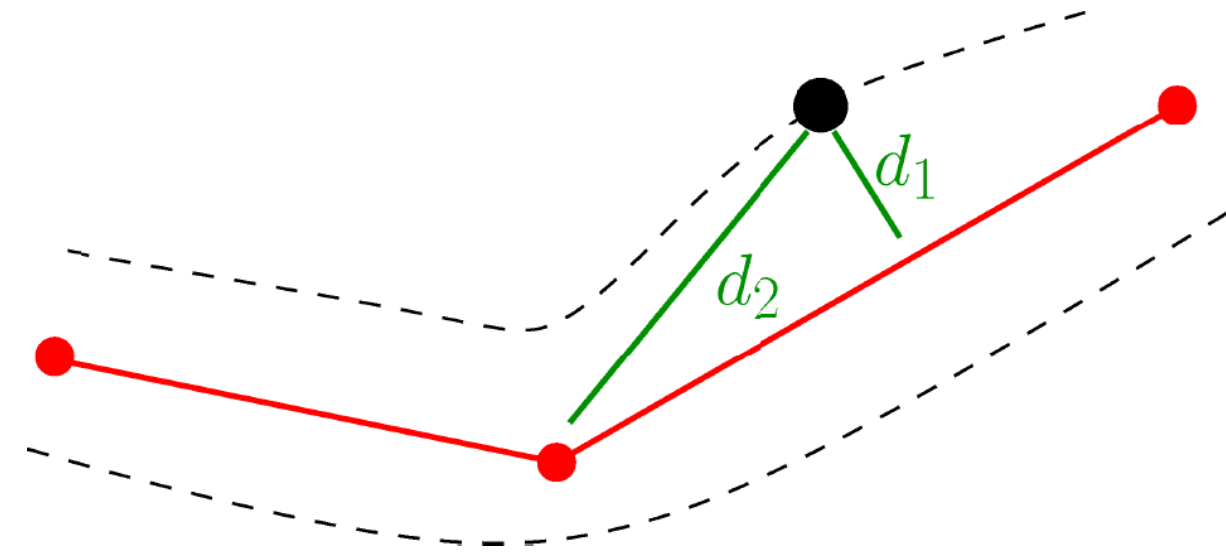
116/139

How to generate skinning weights ?

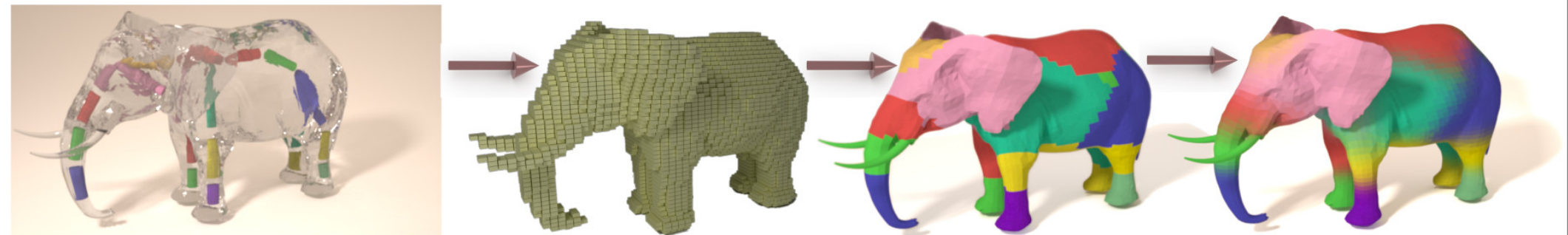
- Paint them manually
- Automatic computation



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
```

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.

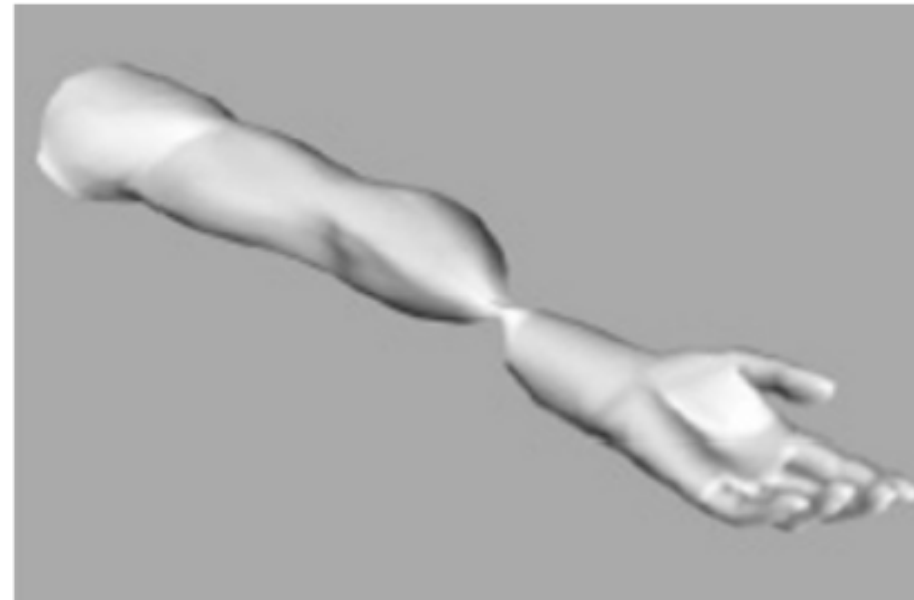
```

.json

Linear Blend Skinning - Limitations

118/139

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

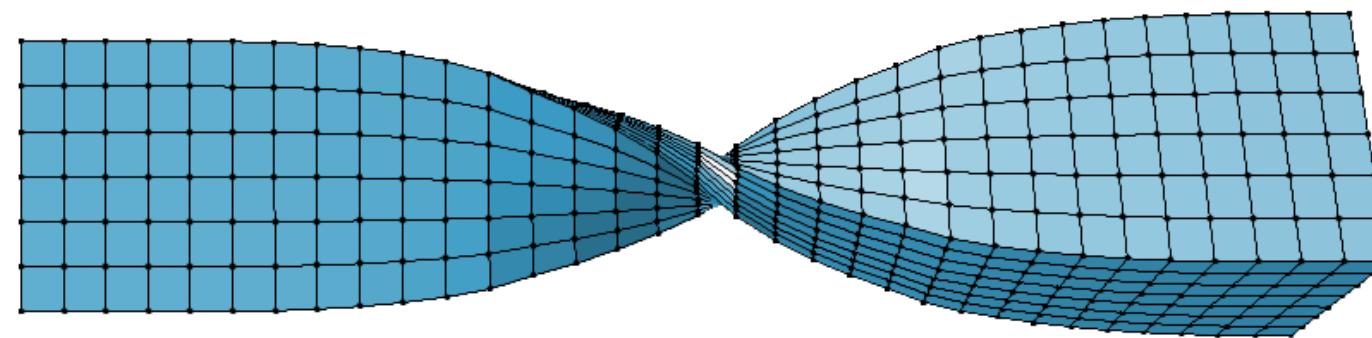


Candy wrapper



Collapsing elbow

Linear blending between rigid transformation matrices



Character Animation

Skeletal Animation

Skeleton structure

120/139

Characteristics

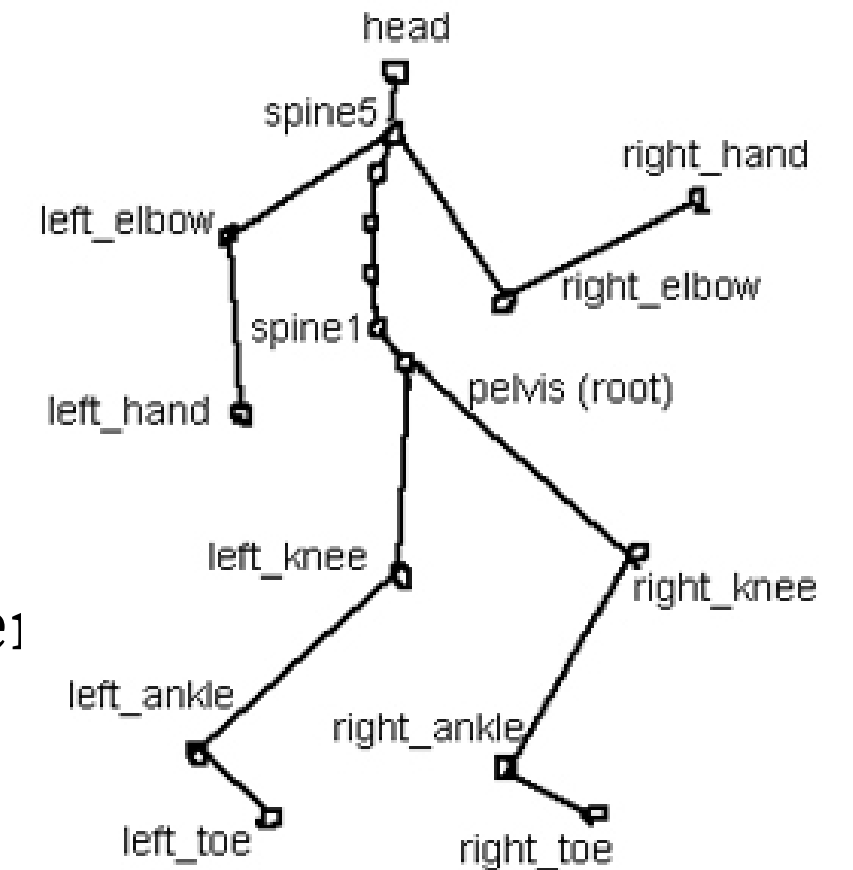
- Hierarchical representation

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

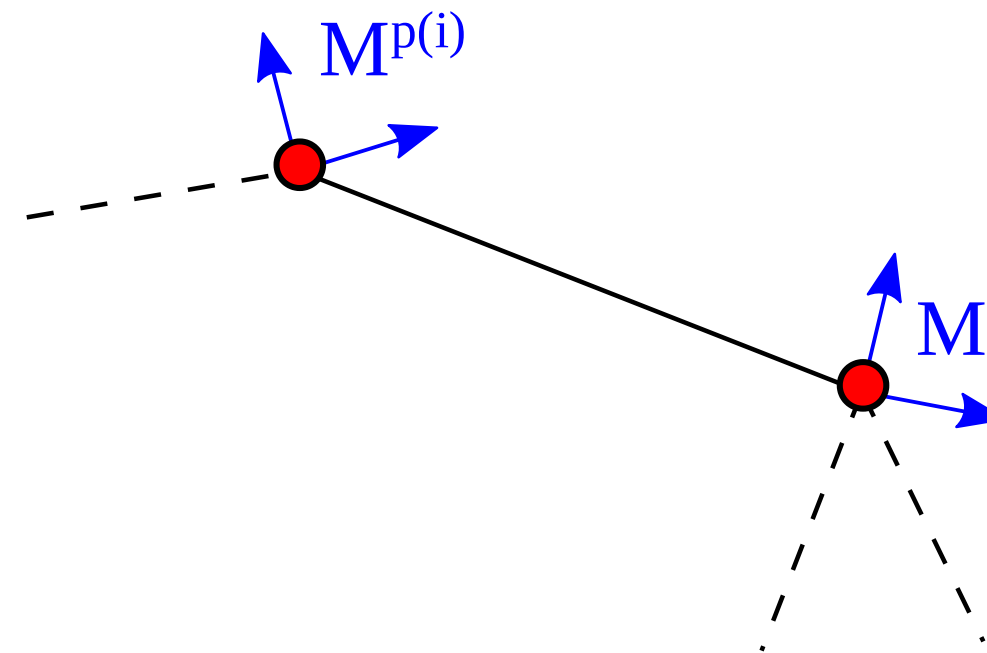
- With 4x4 matrices M

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

- 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

121/139

- 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)
```

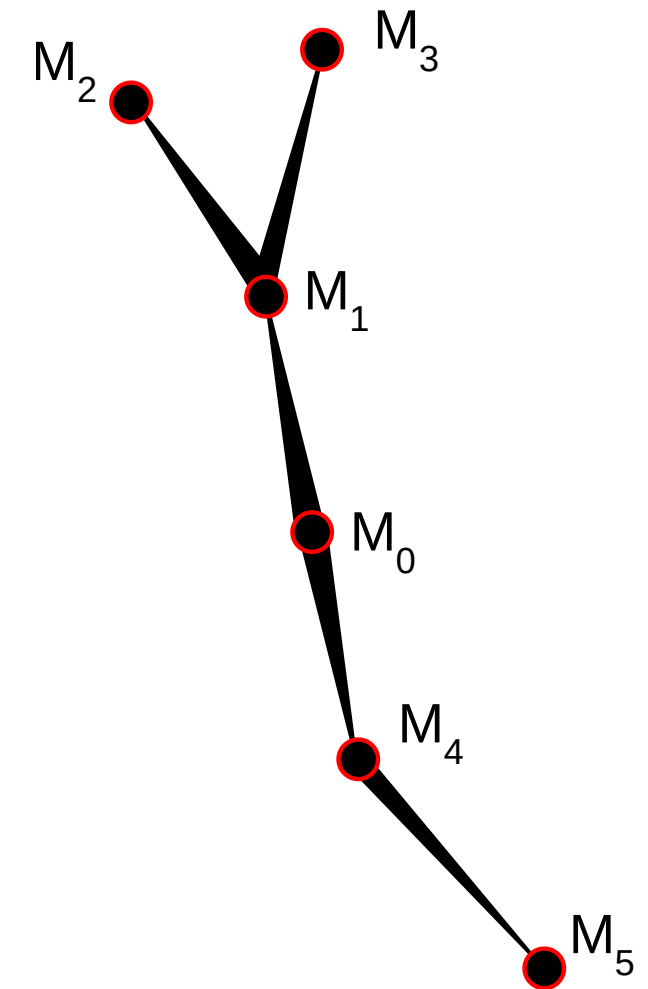
```
{
```

```
    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;
```

```
}
```

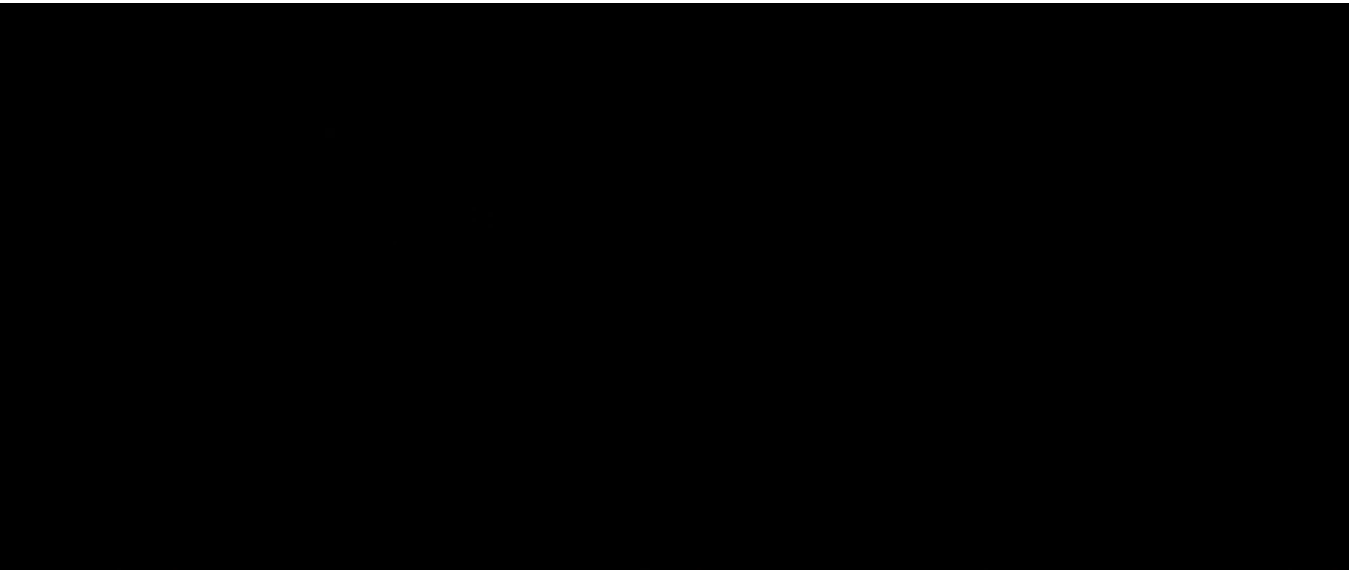
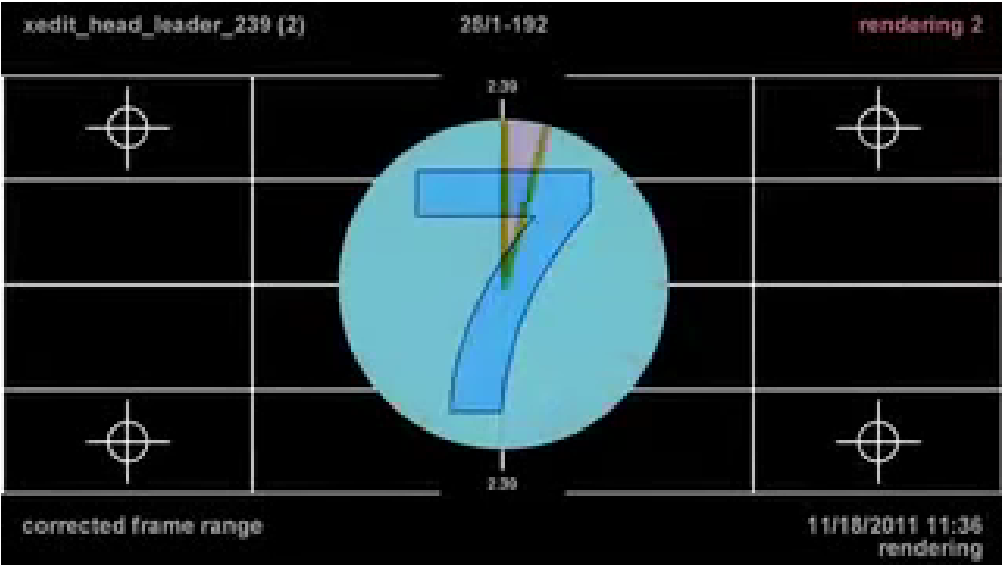
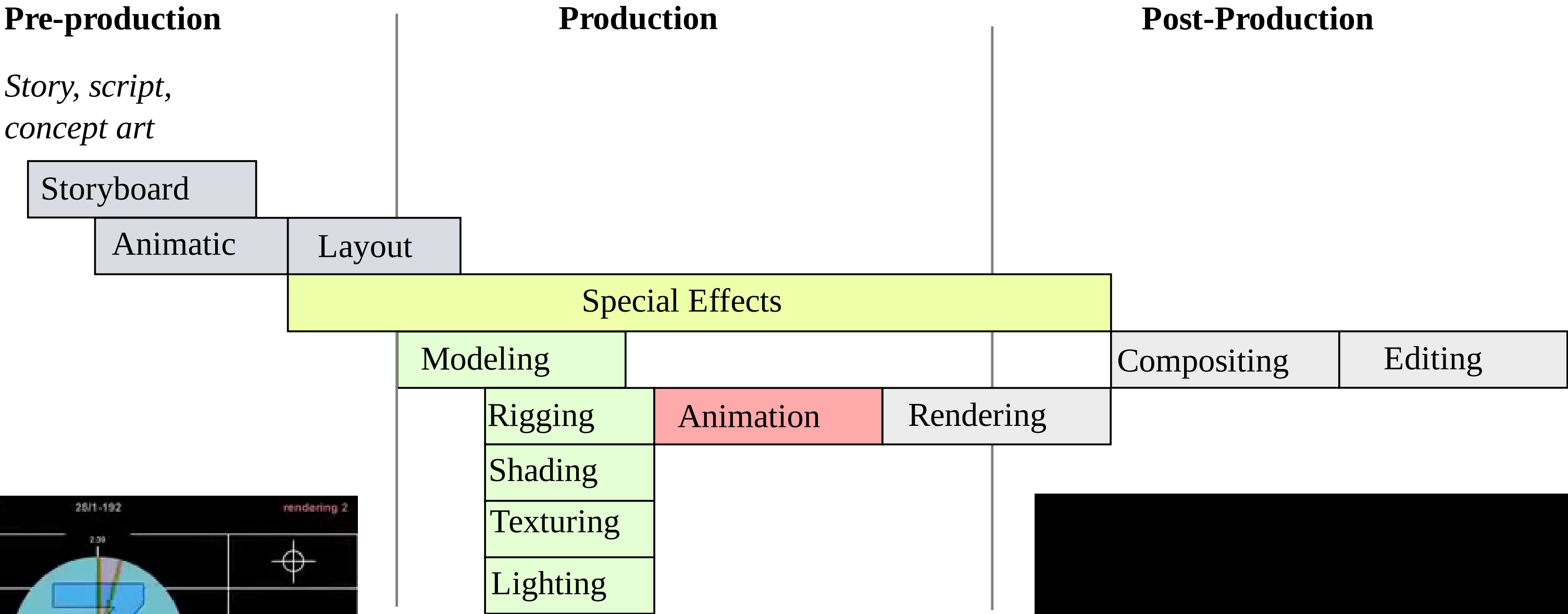


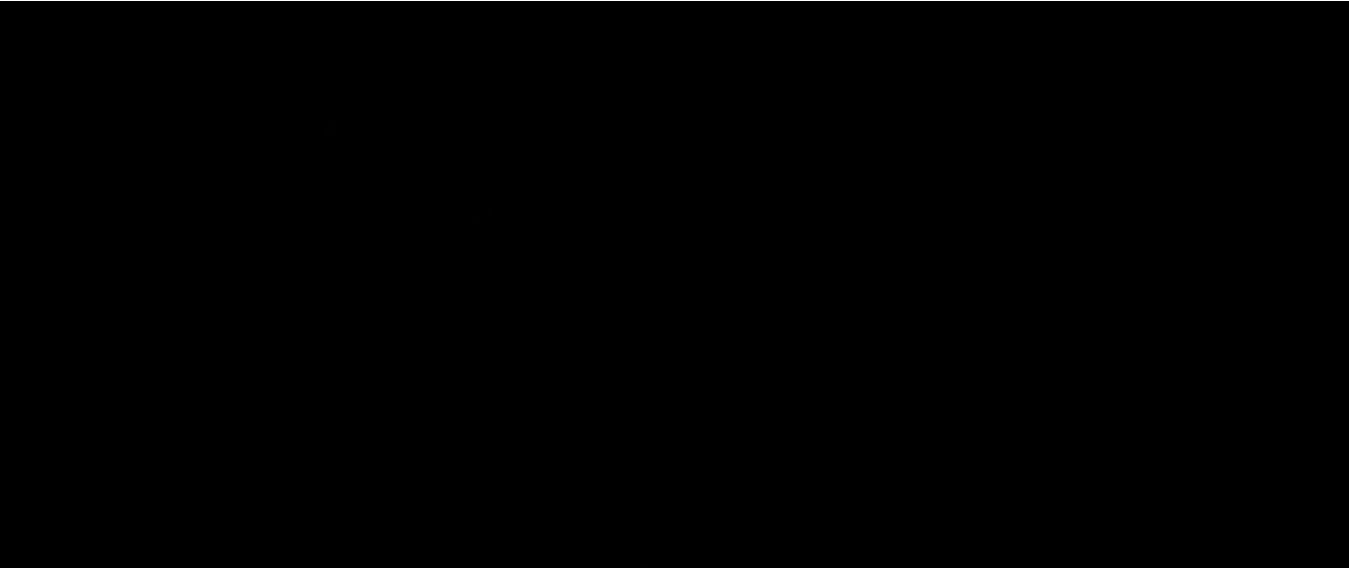
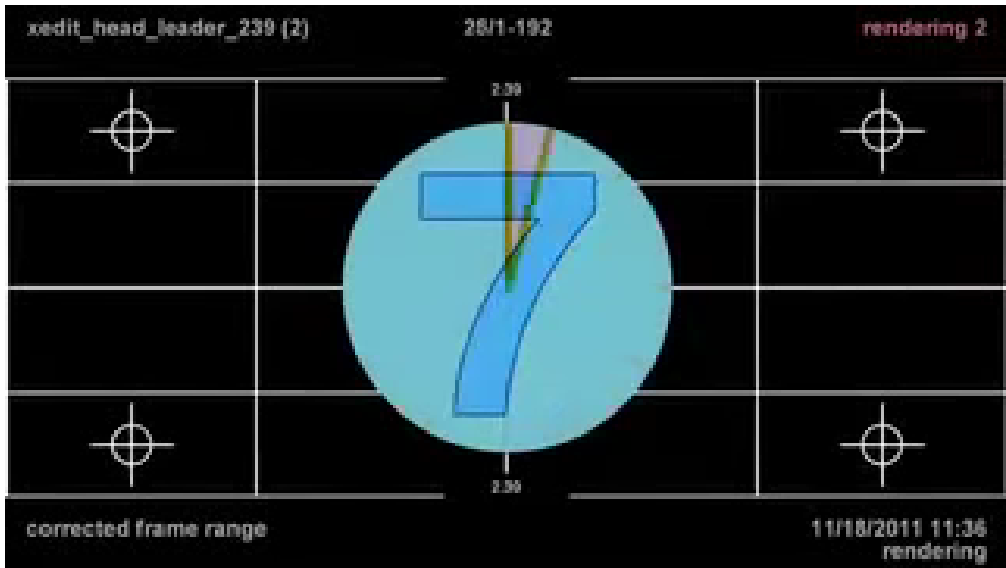
Production Pipeline

Case of Animation Cinema production

Production Pipeline

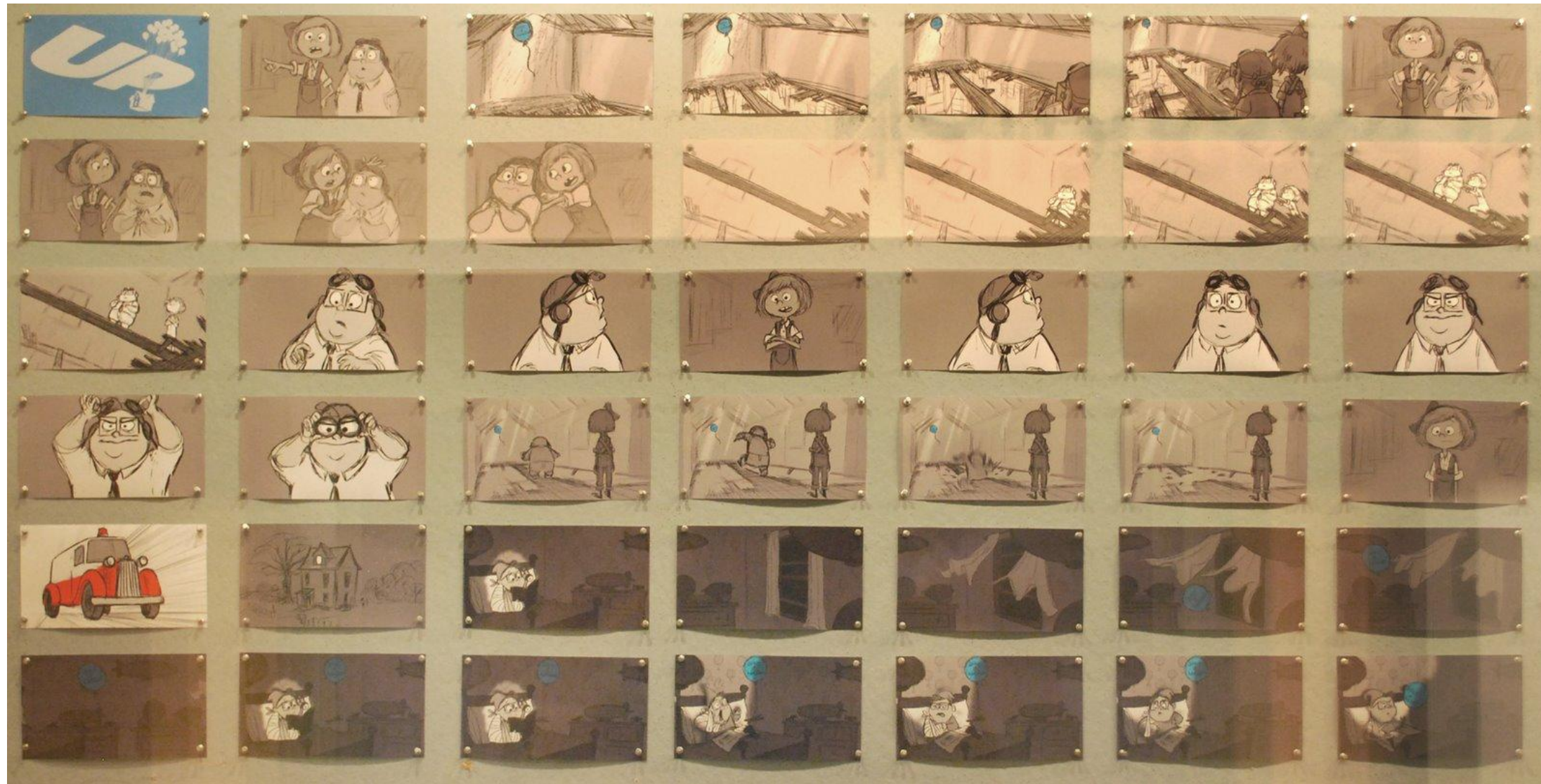
Fundamental structure for VFX/animation movie production.





Story-board

124/139



Few 2D drawings: express the story

Before mostly non-technical - artistic/creative based

Curently Strong increase of Storytelling-related researchs in CG

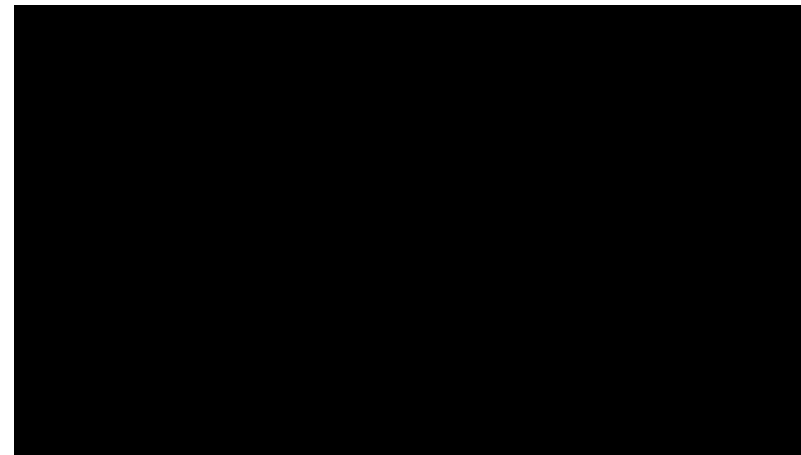
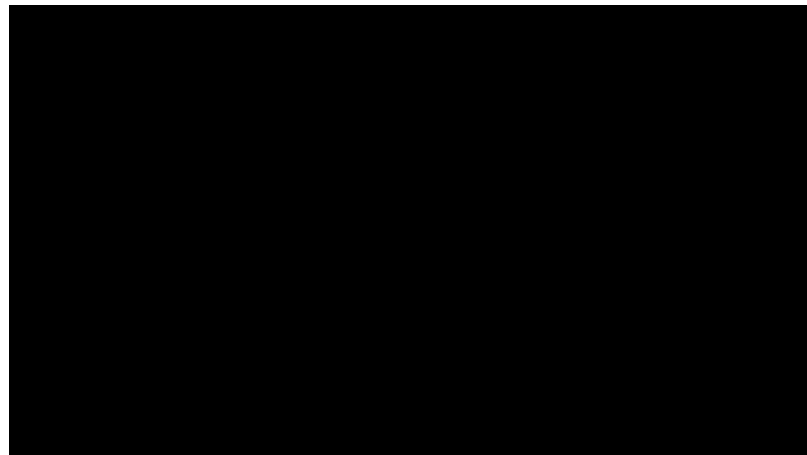
ex. *Disney Research Studios, Anima team at Inria*

Animatic

125/139

≈ Animated story-board, Various format

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



Layout: Moving into 3D

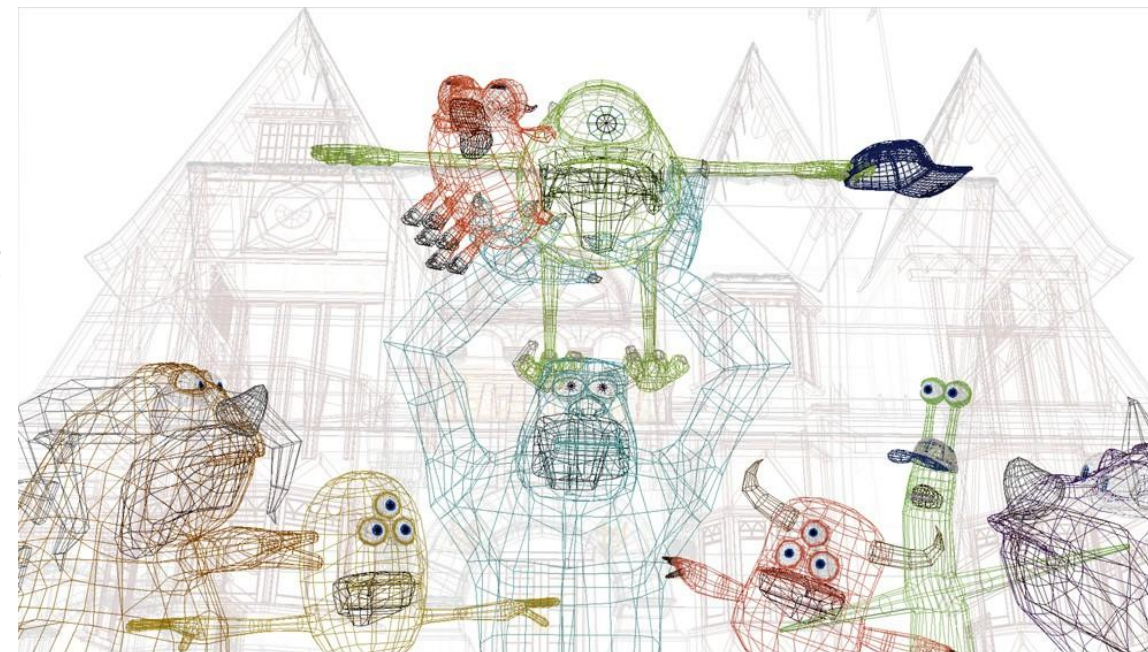
126/139

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

127/139

In production

- Polygonal mesh modeling

Coarse to fine approach

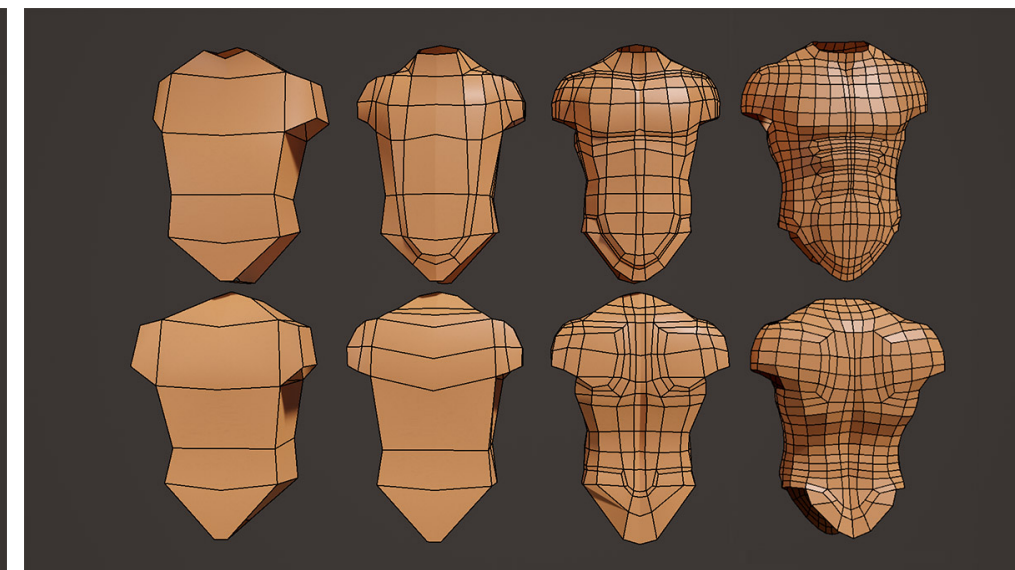
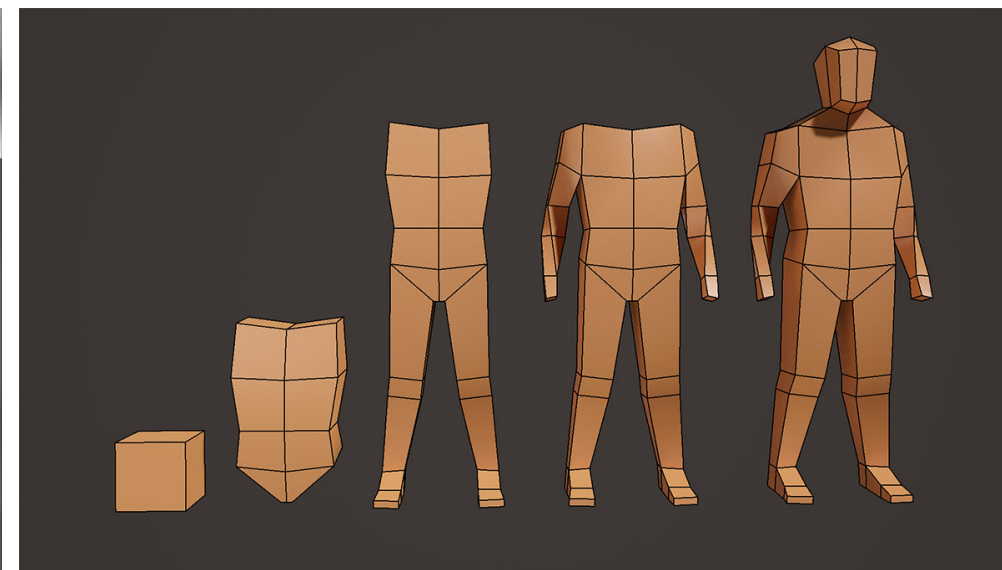
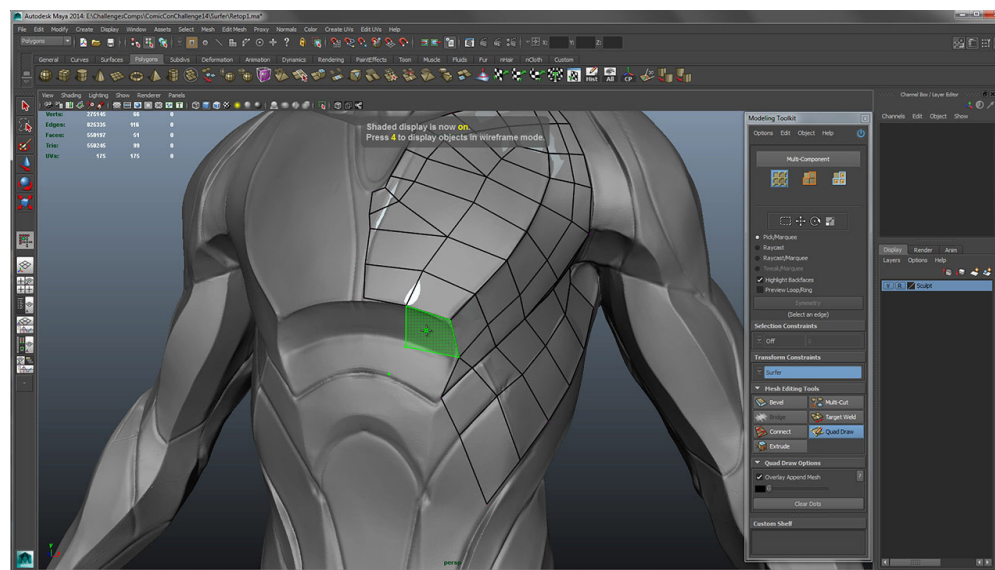
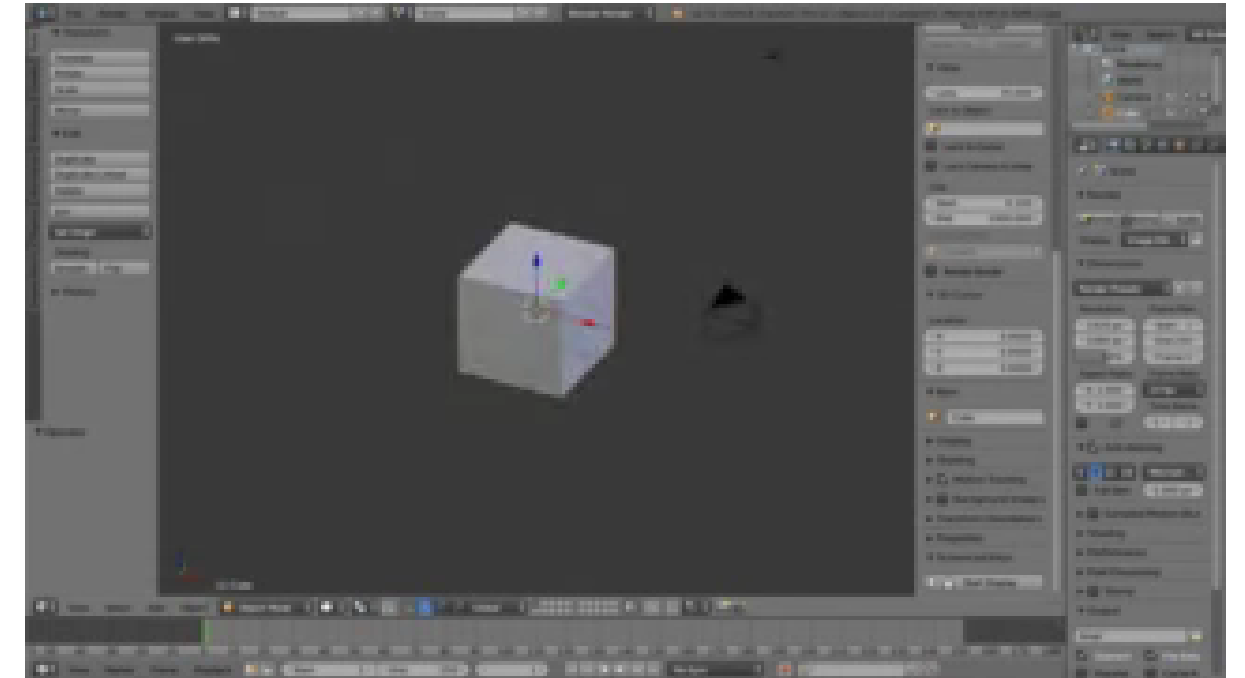
1. Low res modeling (extrusion)

2. Subdivide, Refine

- Parametric (NURBS) modeling

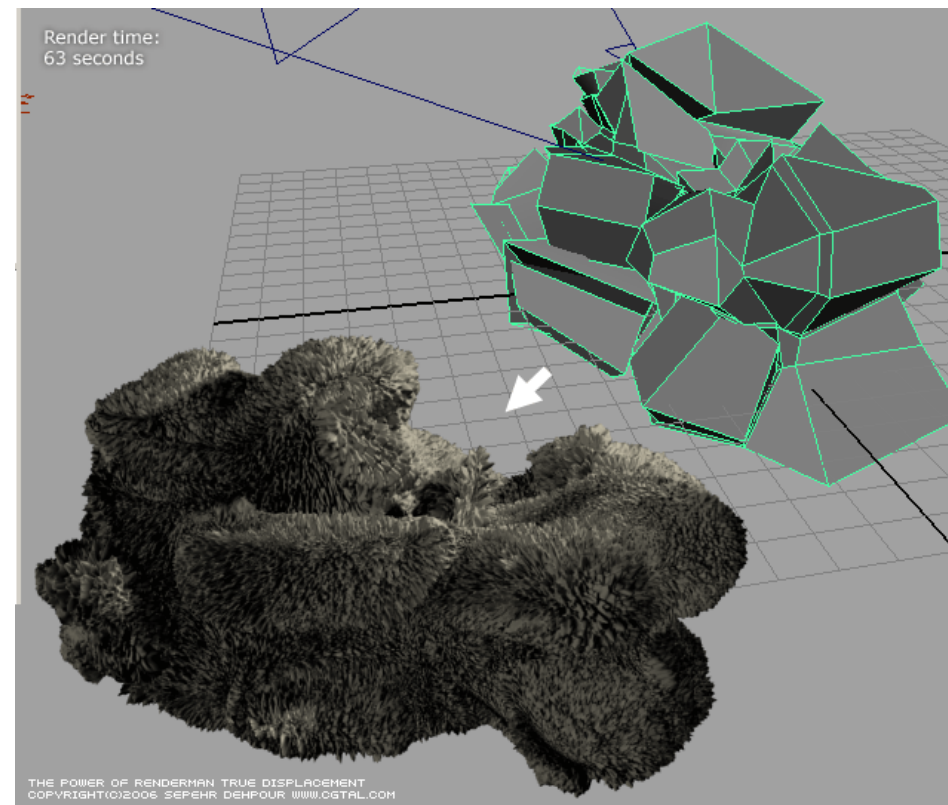
Everything else is "VFX"

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

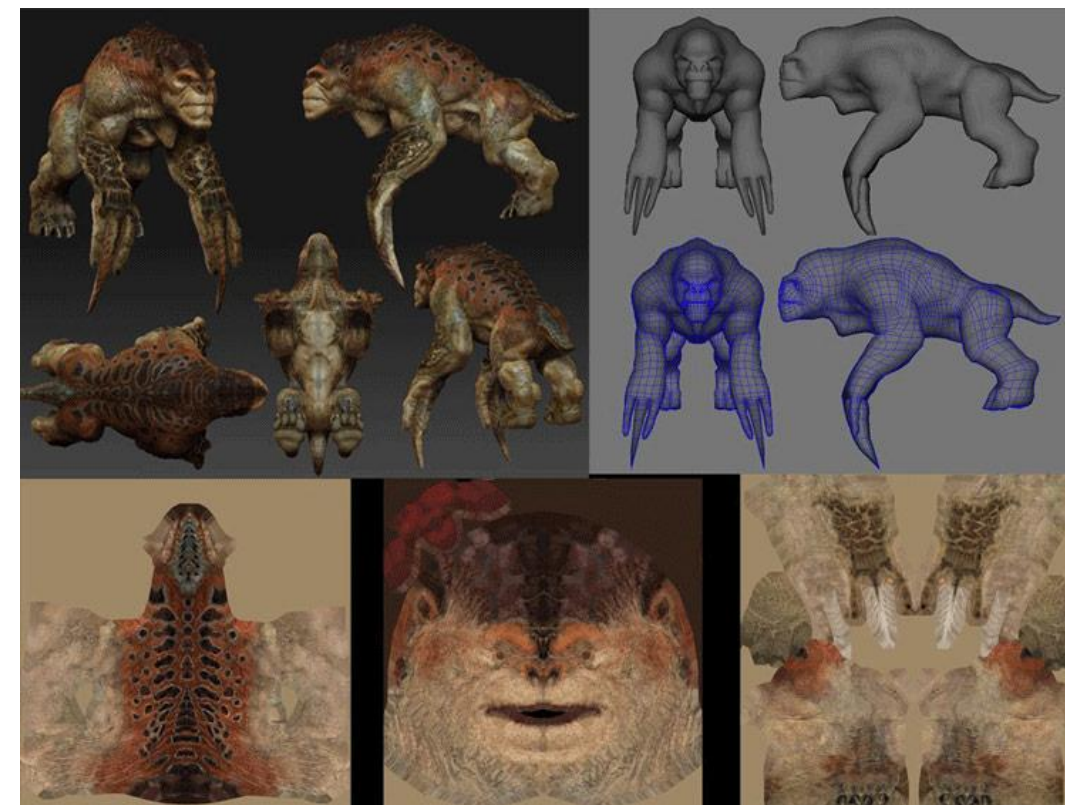


Modeling appearance - Rendering purpose

128/139



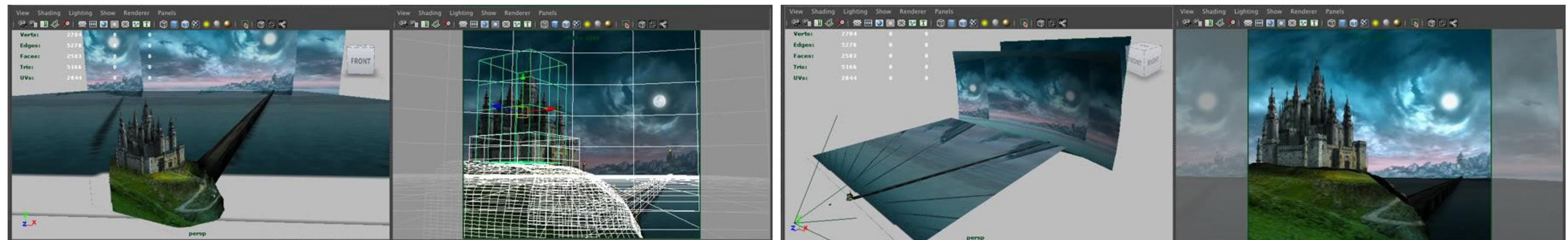
Shading



Texturing



Lighting



Mate painting

Animation

130/139

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*

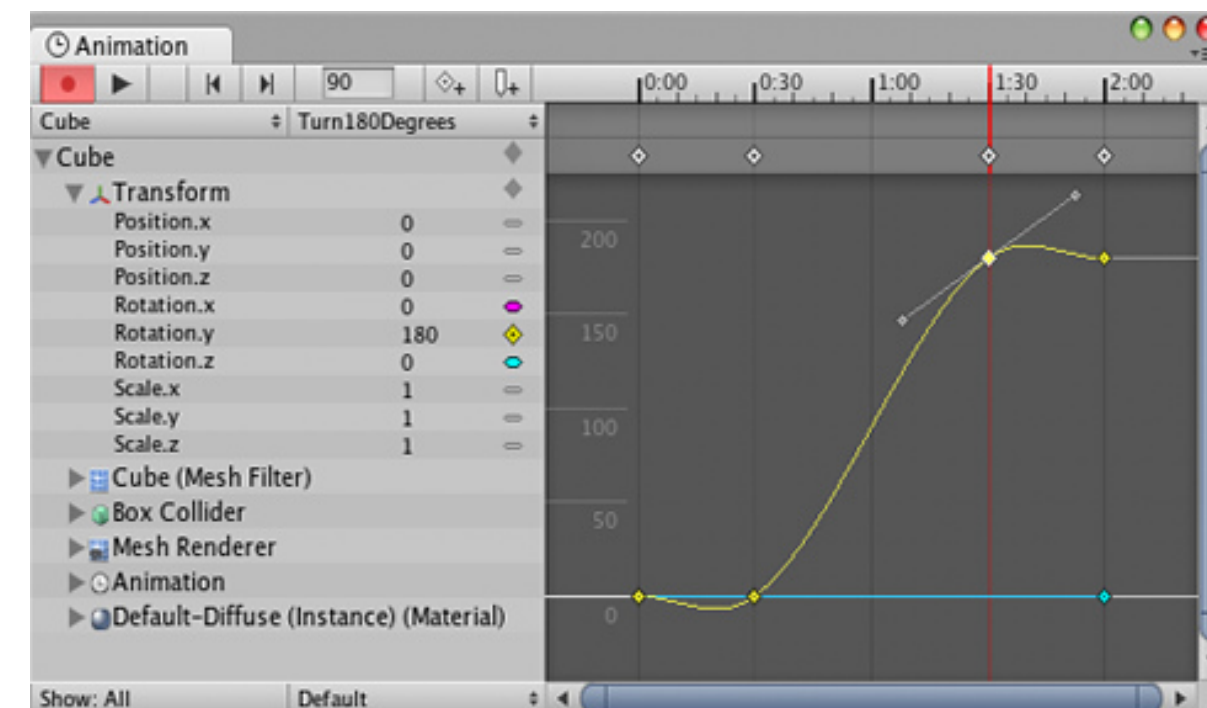
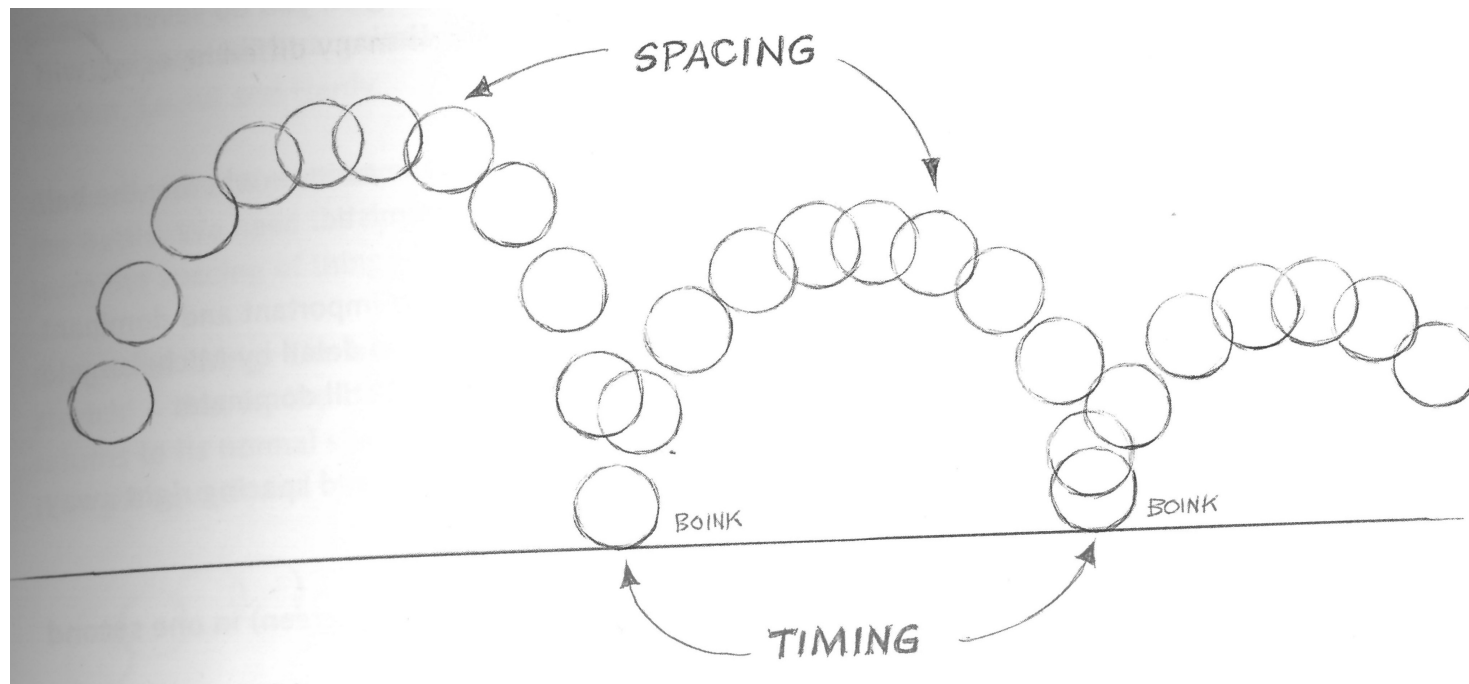
Set the main general posture of the character

*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)

132/139

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

133/139

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

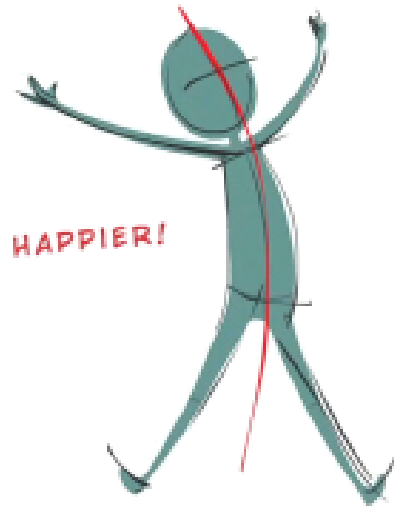
135/139

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

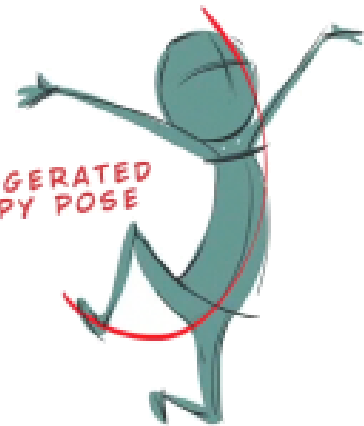
SLIGHT CURVE



HAPPIER!



EXAGGERATED
HAPPY POSE



SLIGHT CURVE

SEEMS LIGHTER



EXAGGERATED EFFORT POSE

SEEMS HEAVIER

ACCENTUATED
CURVE



SLIGHT CURVE



ACCENTUATED
CURVE



EXAGGERATED
SAD POSE



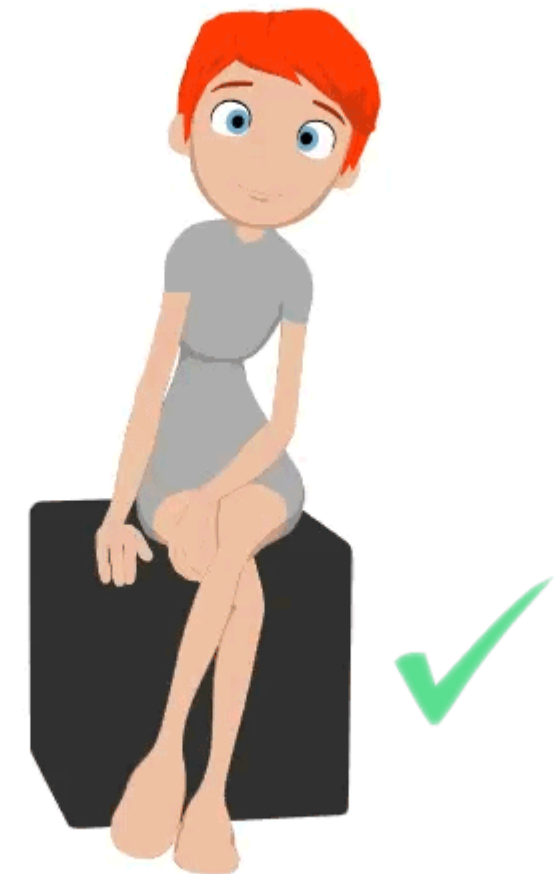
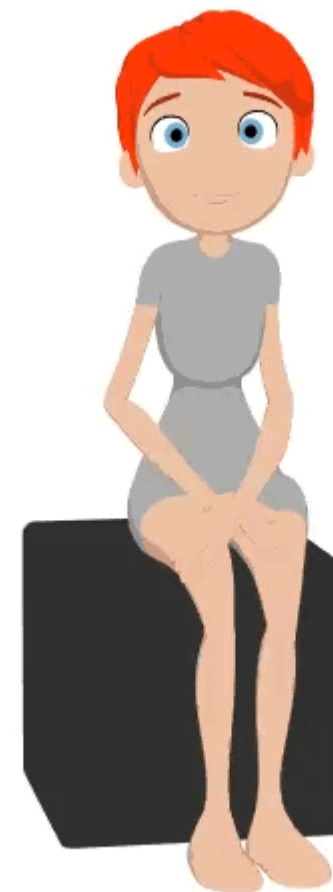
(C) Chiara Porri

Principles of animation

136/139

- 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



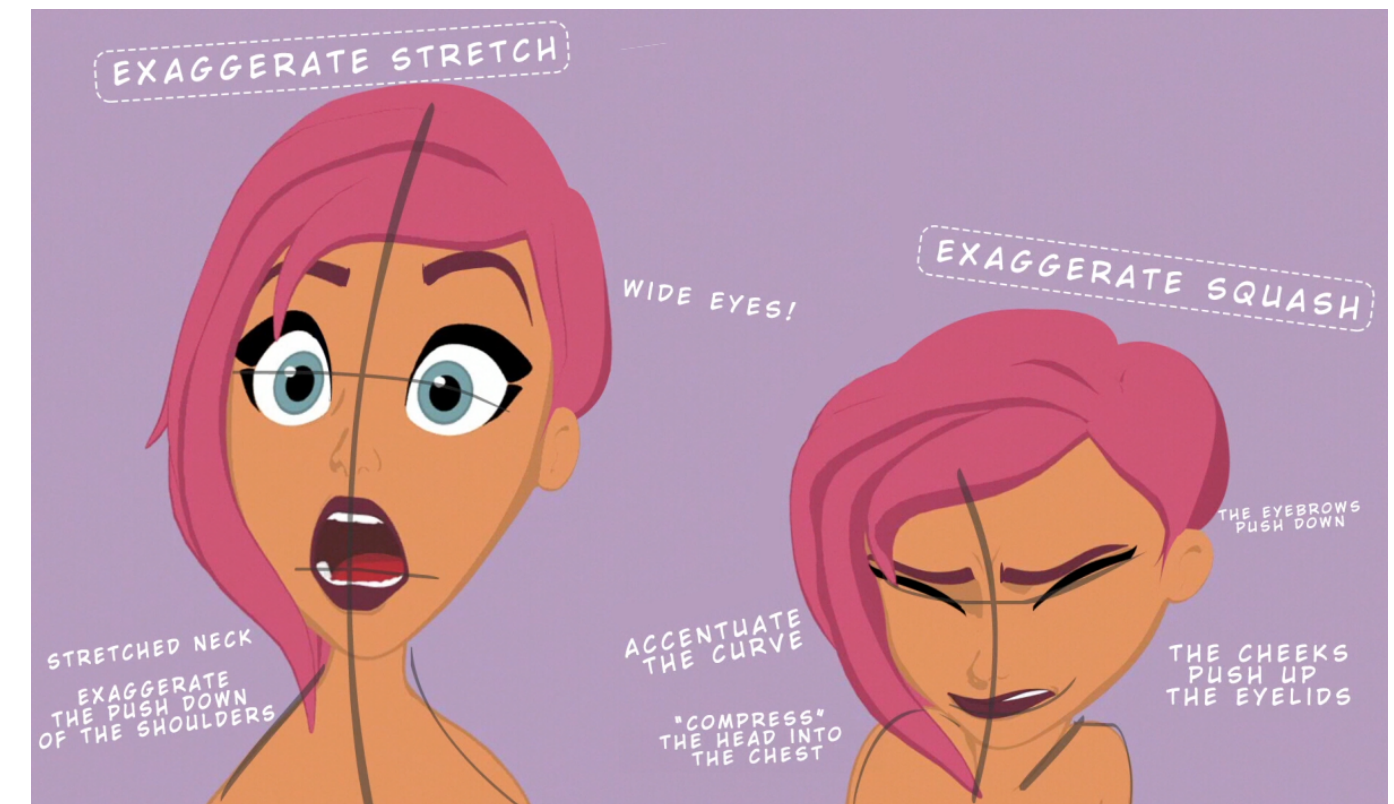
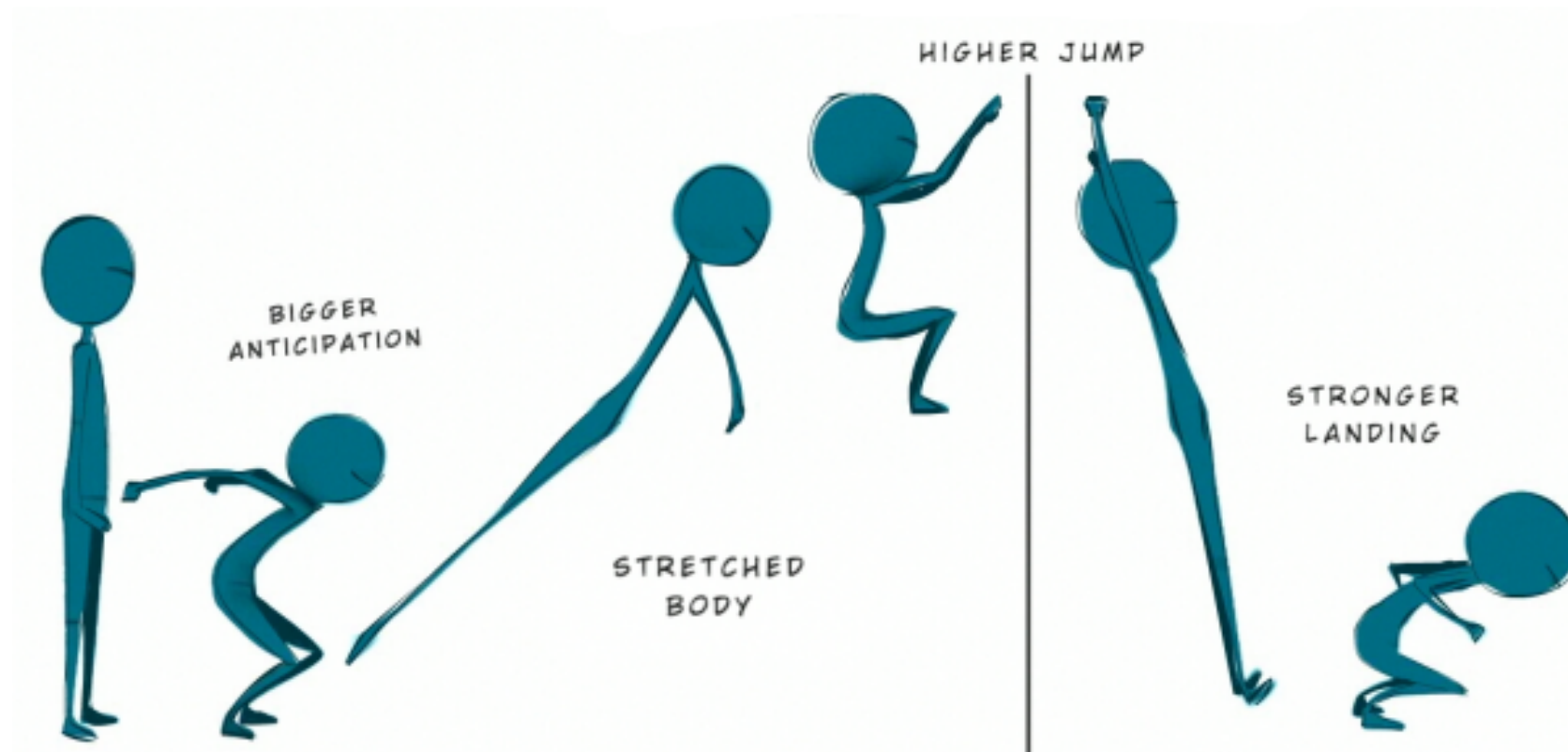
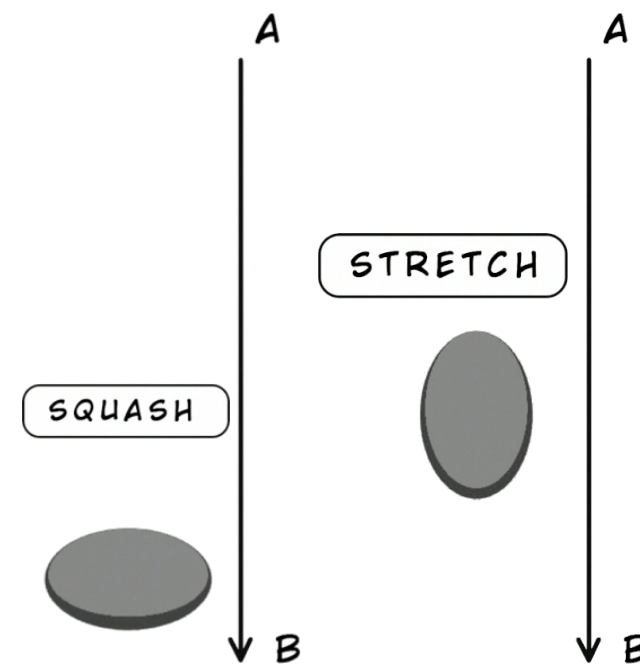
(C) Chiara Porri

Expressive animation

137/139

Squash & Stretch

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

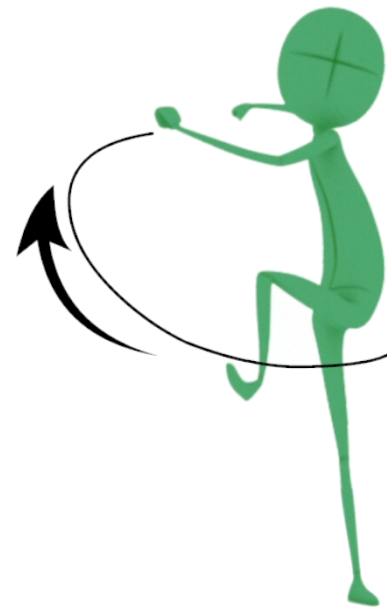


(C) Chiara Porri

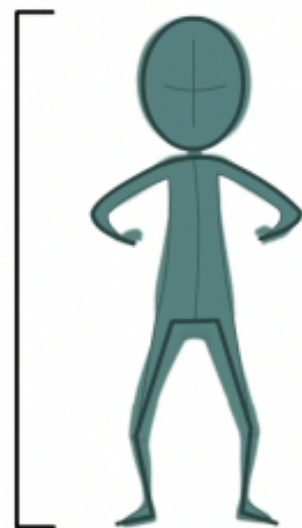
Expressive animation

138/139

Anticipation

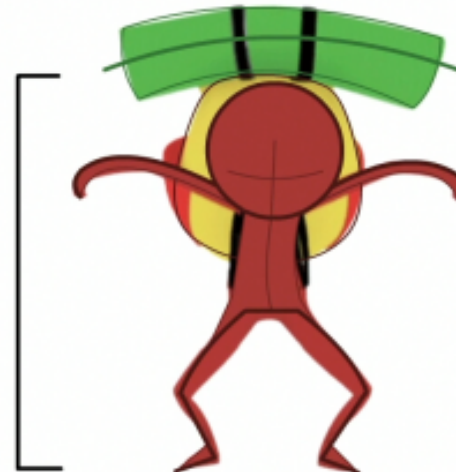


the character is pretty light



Softer Anticipation

the character is bringing
an heavy object



Stronger Anticipation

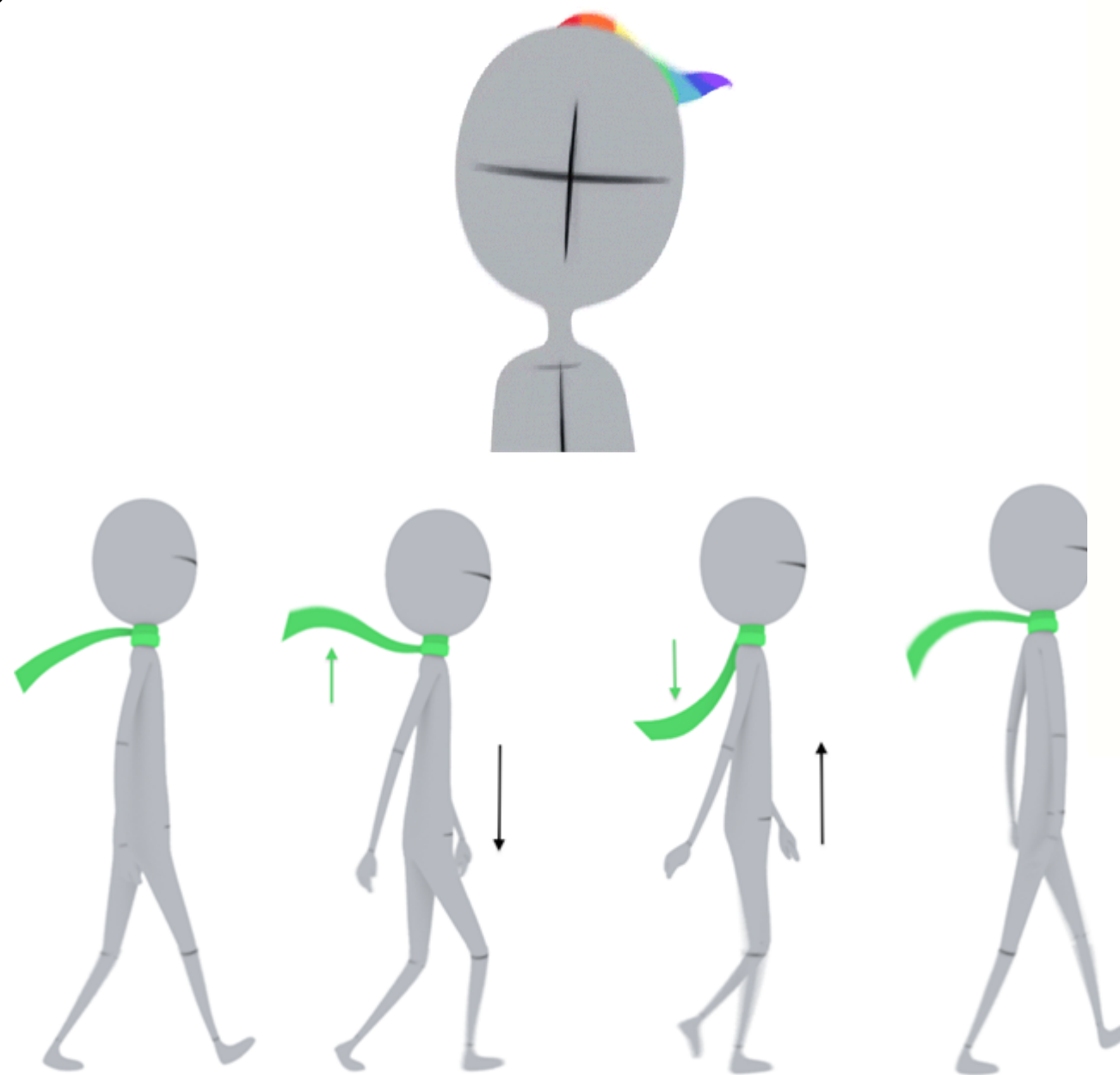
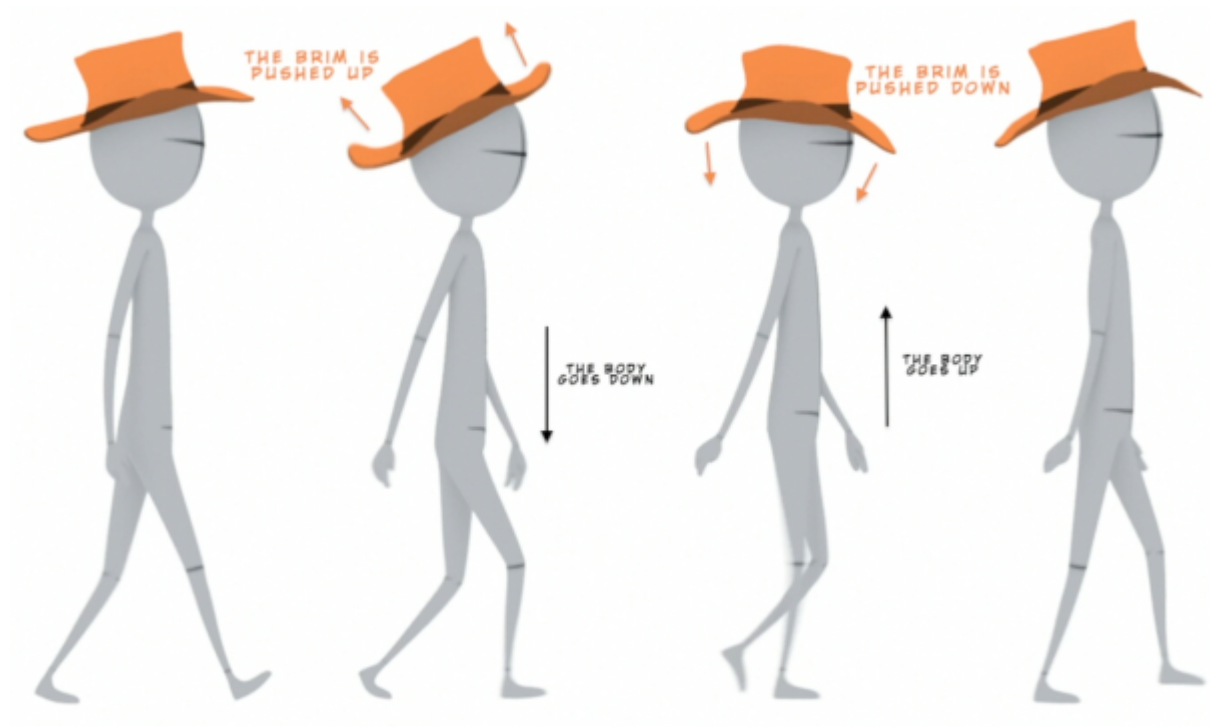


(C) Chiara Porri

Expressive animation

139/139

Follow Through / Secondary motions



(C) Chiara Porri