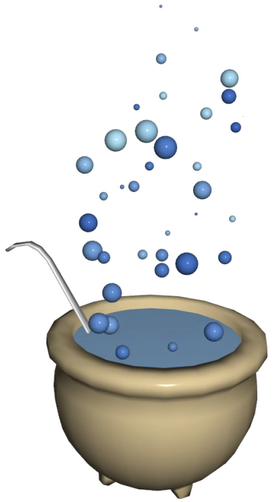
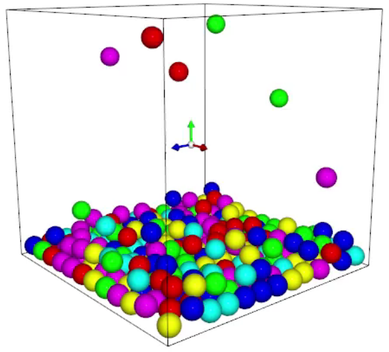
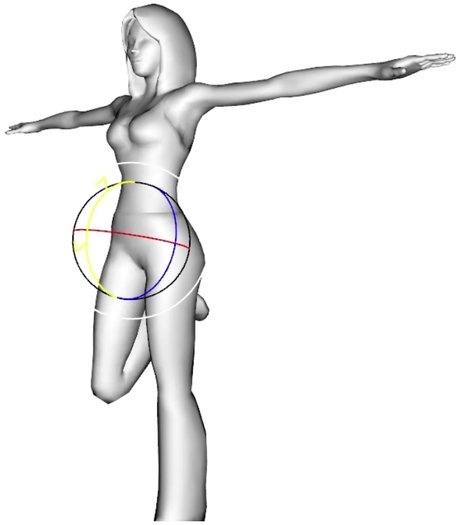
Animation 3D







Damien Rohmer

Damien.Rohmer@polytechnique.edu

EPITA Majeure Image

10/2020

Contexte

Omniprésence des modèles 3D

Application de loisirs

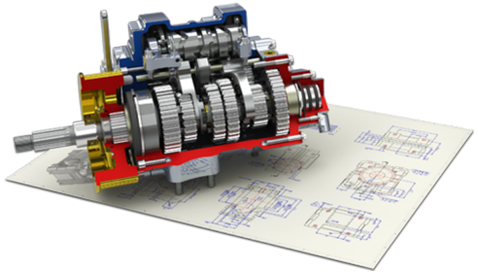
Cinéma/VFX

Jeux vidéos

Réalité virtuelle

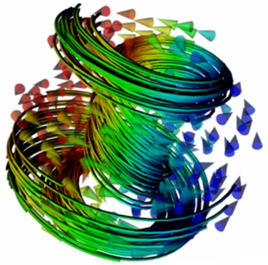
Nombreux autres domaines

CAO



Imagerie médicale

Physique

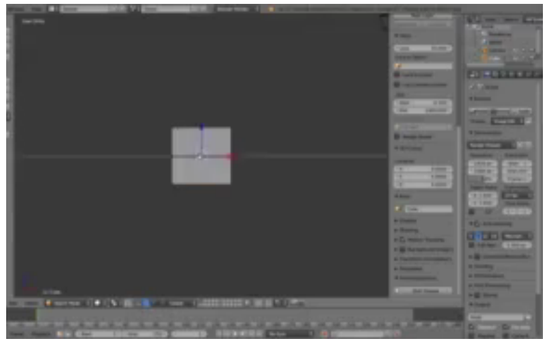


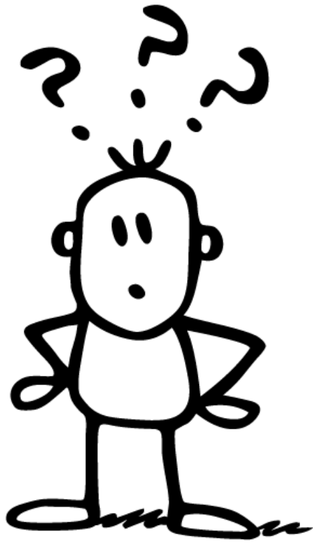
Création 3D

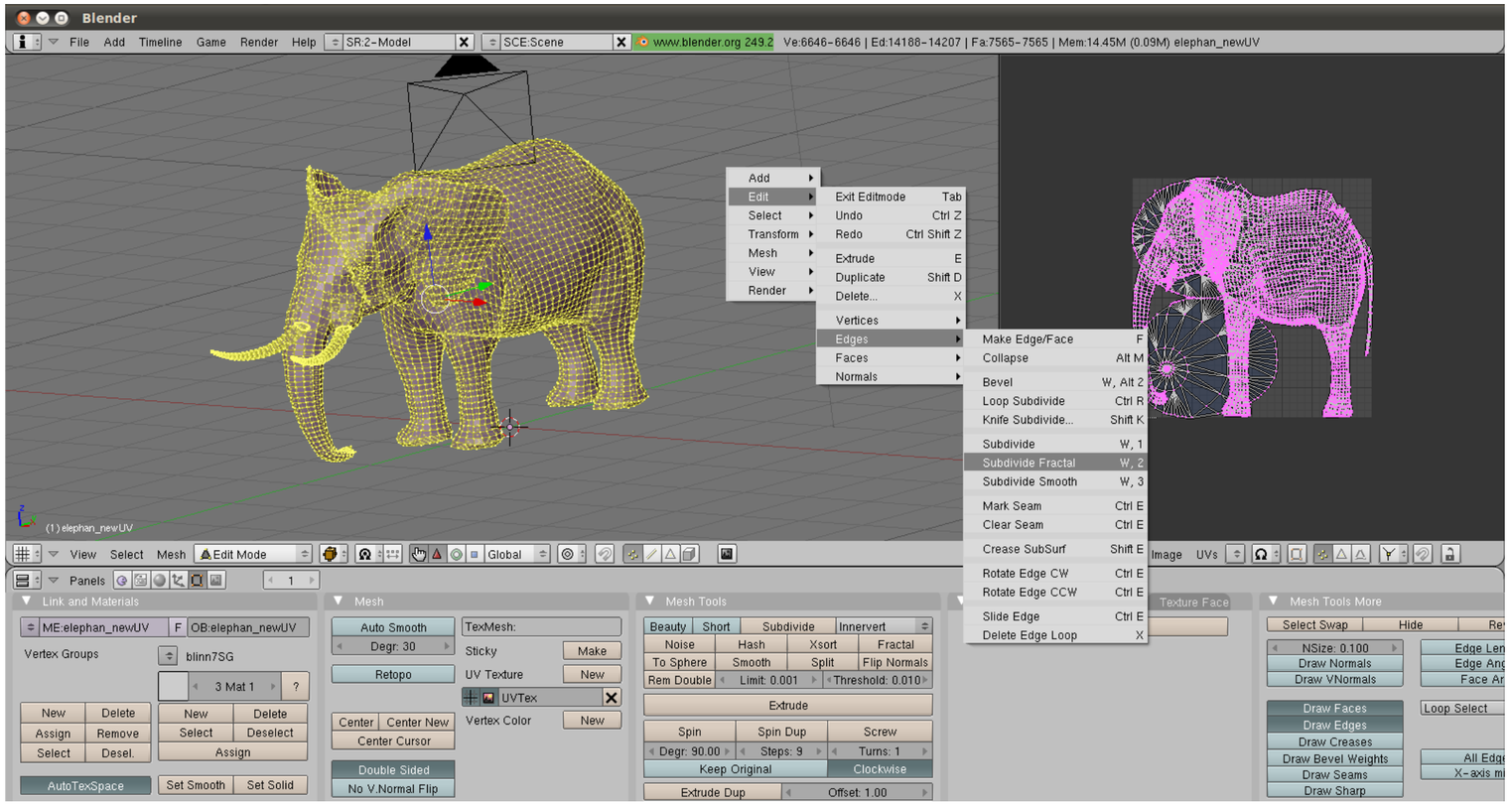
Conclusion: Il est aisé de convevoir et d'animer ses propres modèles 3D.

Création 3D

~~Conclusion: Il est aisé de convoir et d'animer ses propres modèles 3D.~~







Création 3D

~~Conclusion: Il est aisé de convevoir et d'animer ses propres modèles 3D.~~

Cout/temps passé sur la 3D n'a jamais été aussi élevé

- Films animations/VFX
  - *Pirates of the Caribbean ($300M), Avengers ($350M), Toy Story 3 ($200M)*
  - *Cout moyen par séquence VFX (<10s) $50 000*
  - *Cout animation 3D > cout dessin manuelle images à images*

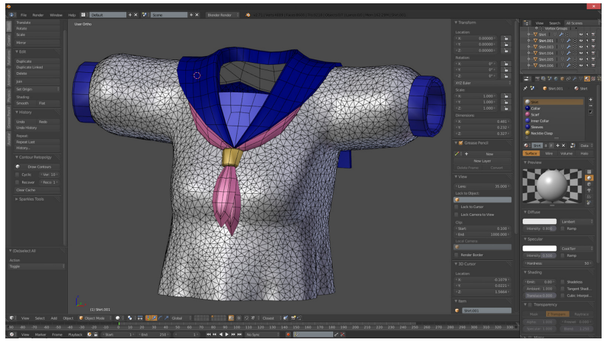
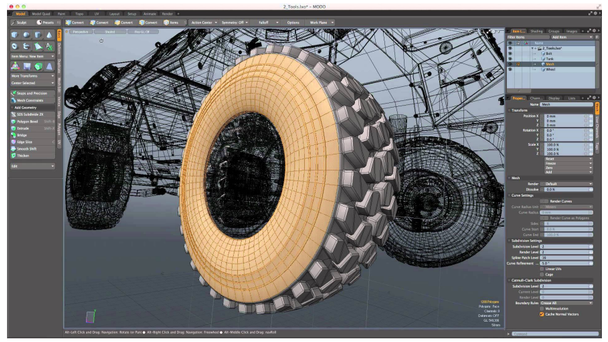
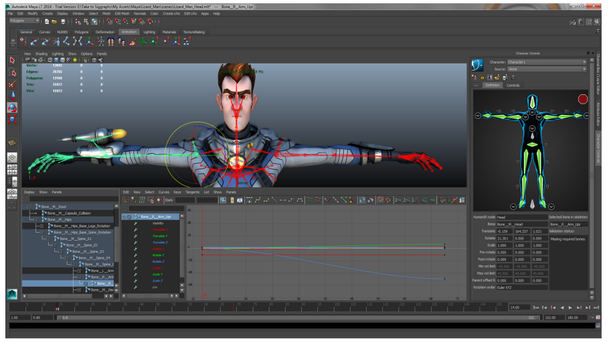
- Jeux vidéos AAA
  $100M, 2 à 4 ans de développements
  >100 développeurs, milliers d'artistes

Création 3D

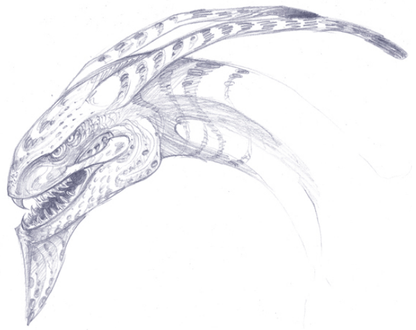
Les outils 3D se sont améliorés

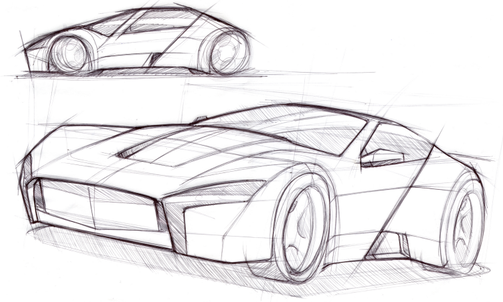
... mais restent complexes et très techniques (3 ans d'études infographistes).

La quantité et la qualité demandé a augmenté plus rapidement que les outils

Dessins/sculpture (à la main) restent plus efficaces pour le prototypage/design






Equipe: Geometric & Visual Computing

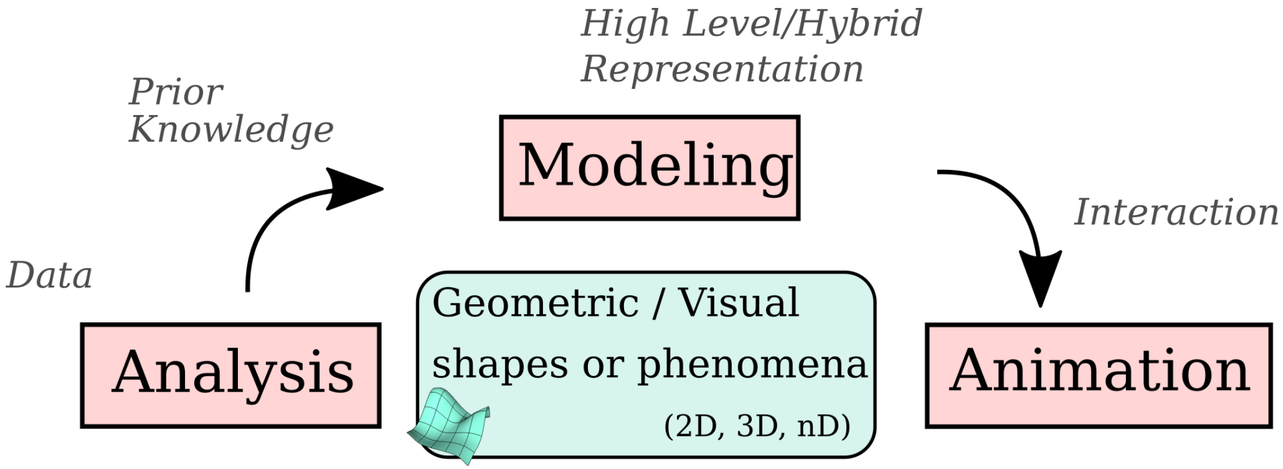
Equipe **Informatique Graphique** et **Vision**

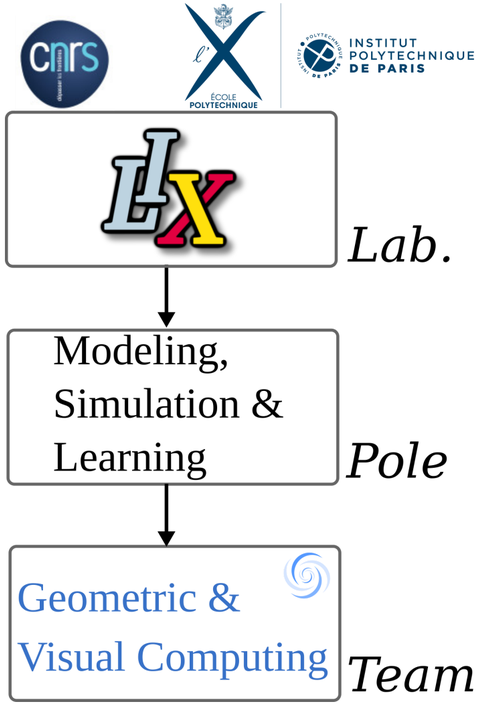
Recherche en **analyse géométrique**, **modélisation** et **animation**.

www.lix.polytechnique.fr/geovic/

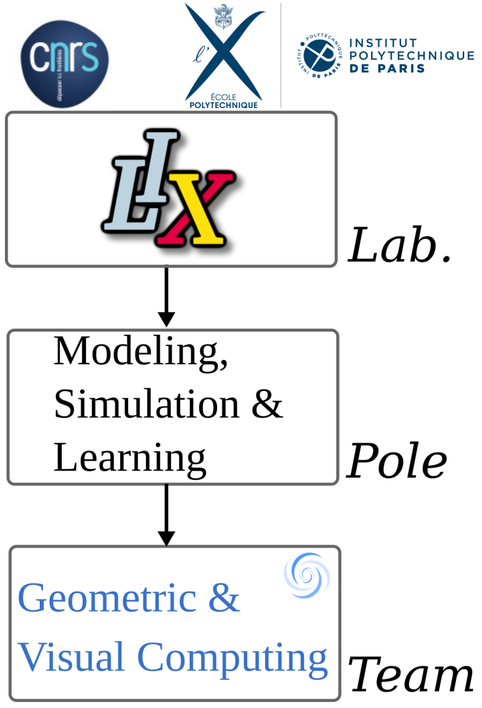
Partie du **LIX** (Laboratoire d'Informatique de l'Ecole Polytechnique), CNRS/X, IP Paris

Pole "**Modélisation, Simulation & Apprentissage**"

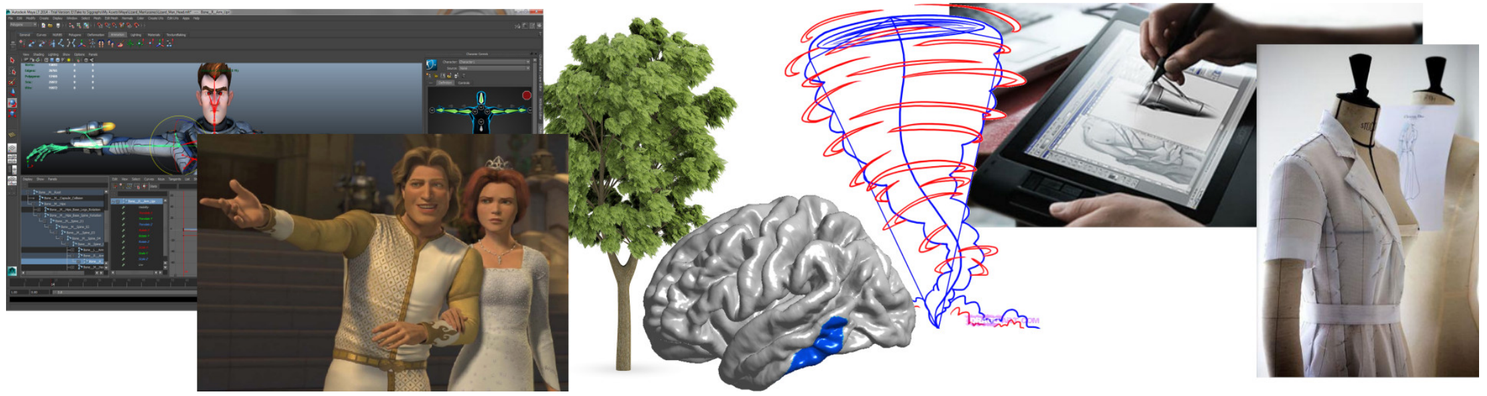




Outils sous-jacents: Geométrie algorithmique, Apprentissage, Optimisation, Modélisation mathématique, etc.

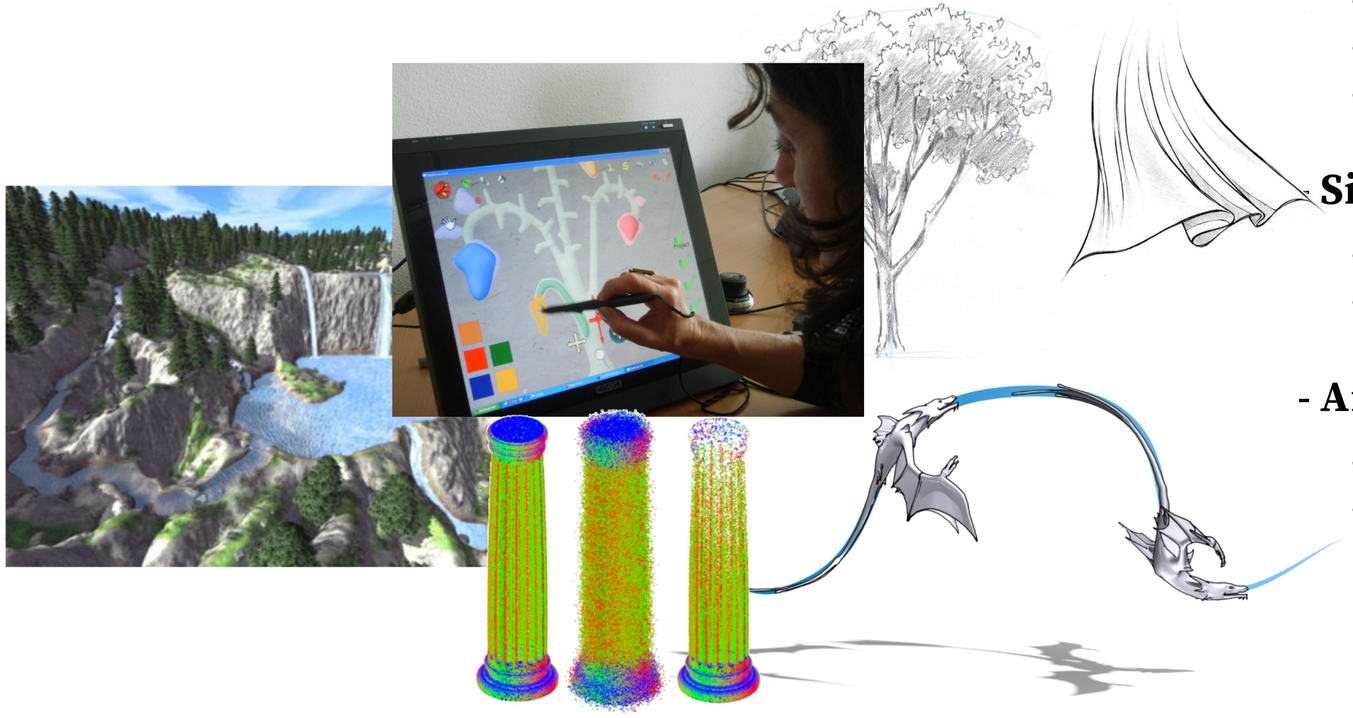


Applications



Domaine d'applications typiques

- **Loisirs & création artistique**
  (Cinéma d'Animation, VFX, Jeu Vidéo)
- **Modélisation & visualisation en Sciences Naturelles**
- **Prototypage et fabrication**



Notre "expertise"

- **Méthode interactive pour l'aide à la créativité**
  - *Modélisation guidés par l'utilisateur: Esquisses, gestes, sculpture*
  - *Contraintes géométriques: Dévelopabilité, volume, etc.*
  - *Design et contrôle d'animation*
- **Simulation Visuelles**
  - *Couplage contraintes géométriques & simulation rapides*
  - *Animation de plis, froissement, détails, etc.*
- **Analyse de formes & algorithmique**
  - *Comparaison de formes et exploration*
  - *Apprentissage sur des données géométriques*

Aide: stages, emploi, poursuite en Informatique Graphique?

Rem. IG: Domaine technique, R&D avancée

- Lien fort sujets recherche & entreprise.
- Thèses IG - sujets appliqués qui intéressent les industries.

Si le domaine vous intéresse:

AFIG - Association Française d'Informatique Graphique

https://www.asso-afig.fr/site/

Listing entreprises & *centre de recherches*.

Listing offres stages, thèse, ingénieurs.

+ N'hésitez pas à contacter les chercheurs.

Animation 3D - Plan du cours

3 cours (matin) de 2h

3 TP (après-midi) de 4h

1. **Introduction et rappels d'Informatique Graphique**
2. **Warm-up système de particles**
3. **Animation descriptive**
   - Pipe-line de production 3D, animation expressive
   - KeyFraming, interpolation de positions
4. **Animation physique**
   - Application simulation de particules
   - Simulation de tissus
5. **Animation de personnages**
   - Déformation de la peau: Skinning
   - Squelette d'animation: cinématique directe/inverse

Evaluation

A confirmer

- Un controle continu
  - un petit questionnaire papier $\simeq 15$ min le 16/12/2019
  - sans documents ?
- Un compte rendu de TP
  - Collision de sphères, tissus, ou personnage articulé
  - $\simeq$ 5 pages

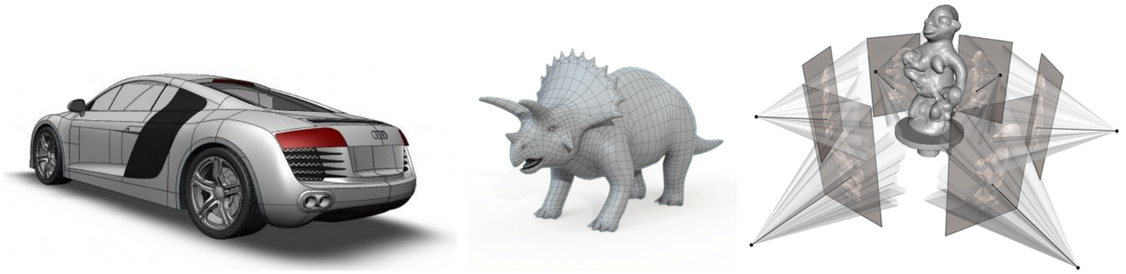
    Votre démarche, vos résultats, vos ***analyses***

Computer Graphics notions and CG programming

Computer Graphics main SubFields

Modeling

How to create static shapes



Animation

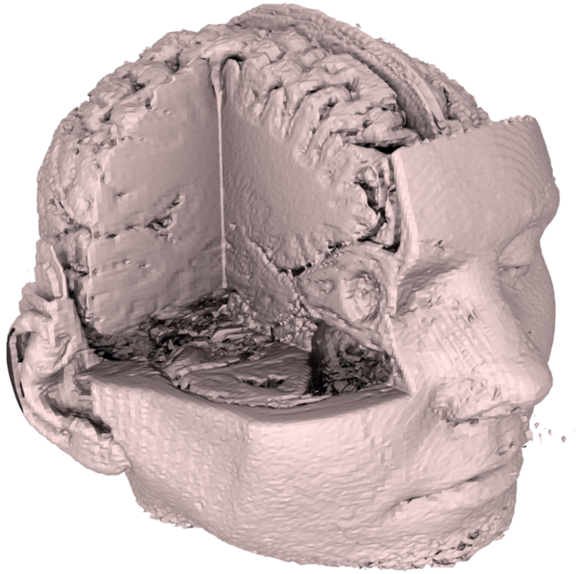
How to create and author time varying shapes

Rendering

How to generate 2D images from 3D data

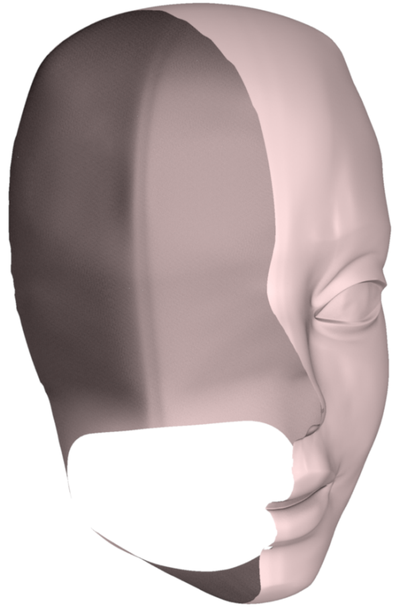
Representing 3D shapes for Graphics Applications

Volume representation



+ Accurate, handle density

Surface representation



+ Focus on visible part

+ Fast GPU rendering, low memory footprint

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

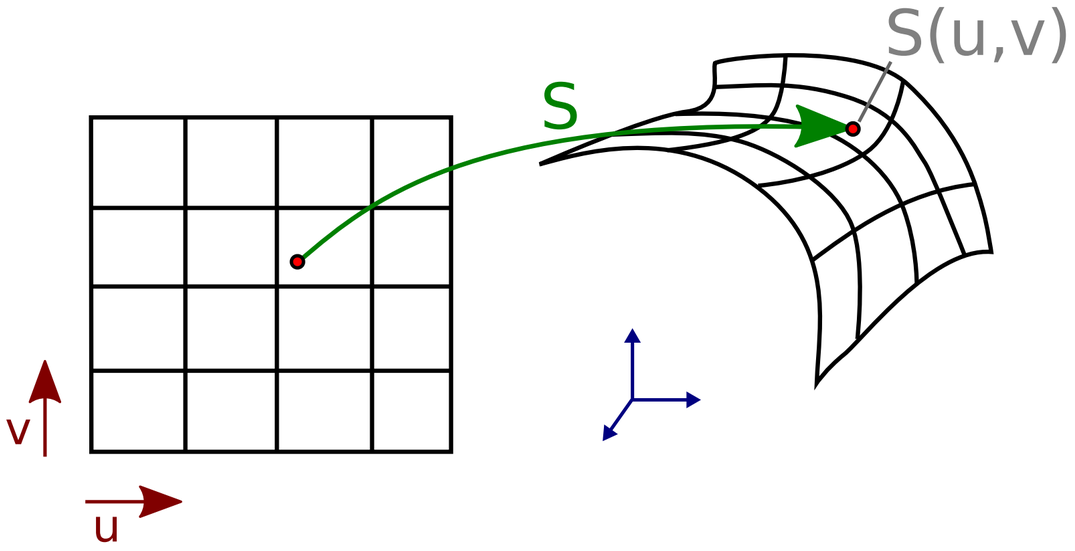
=> **Scientific visualization**: Volume data

Two main representations for surfaces

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

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

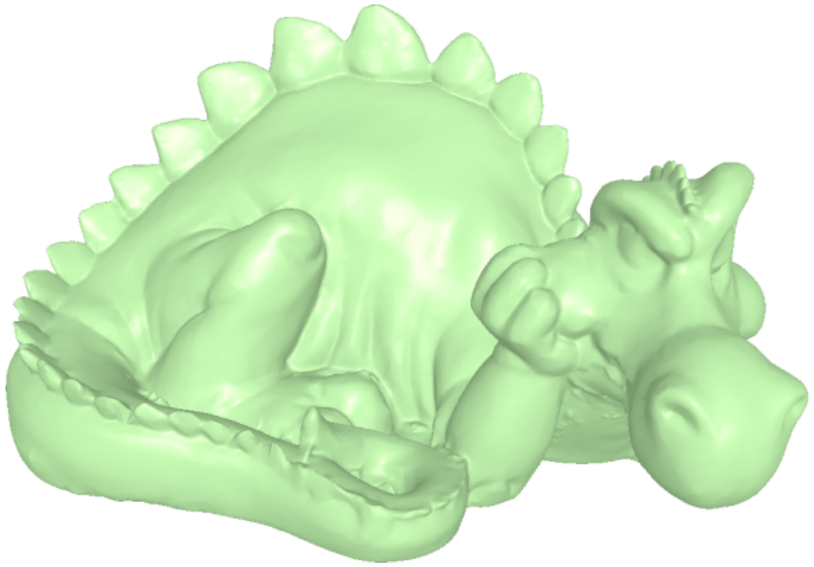
$$S = \{(x,y,z) \in \mathbb{R}^3 \mid F(x,y,z) = 0\}$$

Isosurface of scalar field

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

Difficulty of surface representation using function

Which function can represent this shape ?



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

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

Objective of surface representation

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

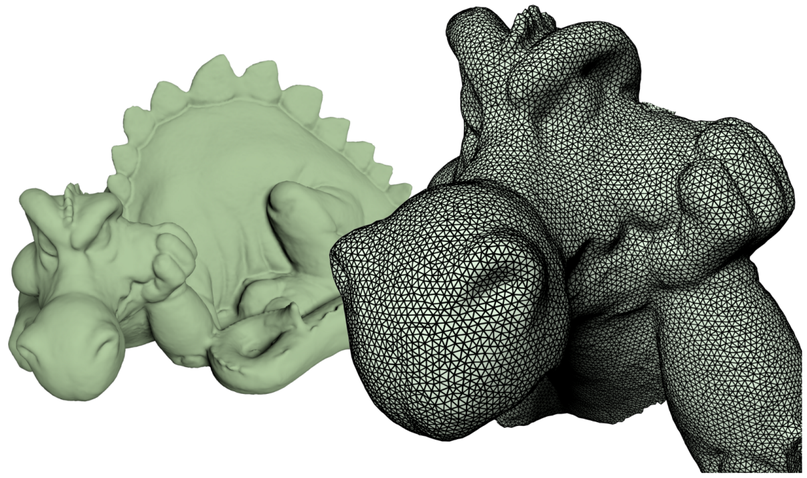
+ Simplest representation

+ Fit to GPU Graphics render pipeline

- Requires large number of samples: complex modeling

- Tangential discontinuities at edges

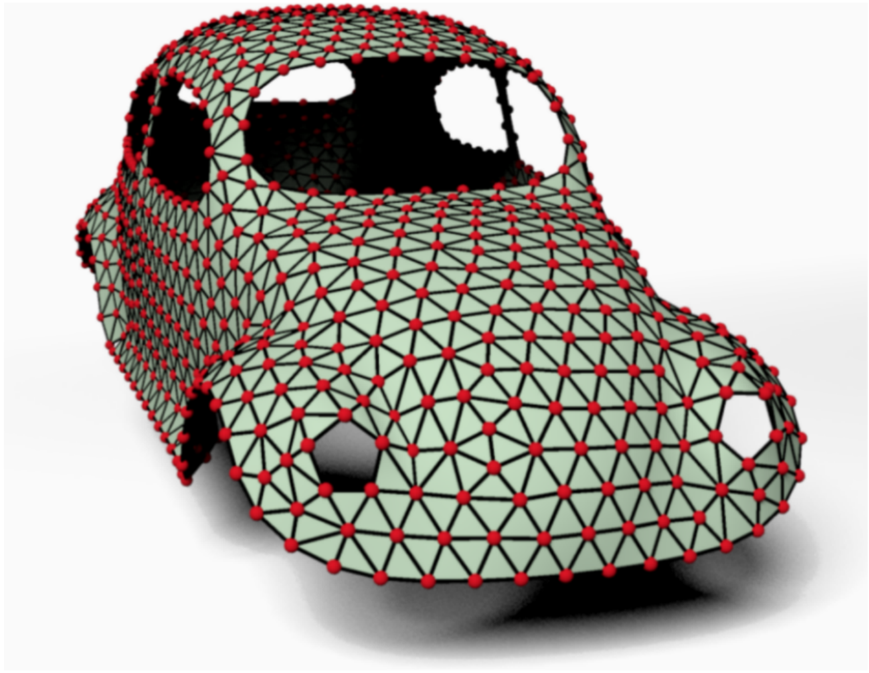
Meshes



Simplest possible representation of 3D surfaces: set of triangles

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

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



● Vertex

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

／ Edge

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

◢ Face

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

Mesh encoding

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)]
```

=> Prefered solution

+ more space efficient

+ modifying 1 vertex = 1 operation

Mesh buffer encoding in C++

```cpp
#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;
}
```

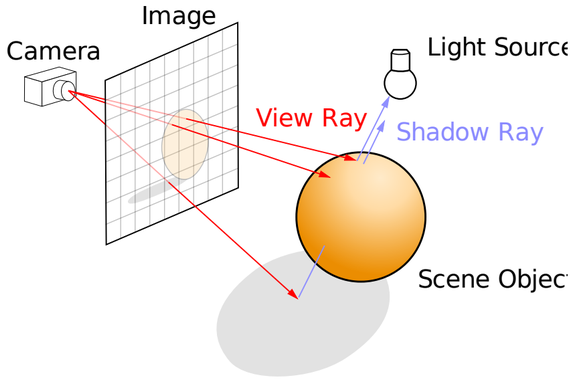
Example of 3D Mesh file

```
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
```

Open question: How to display it efficiently on screen?

How to render surfaces

Ray tracing

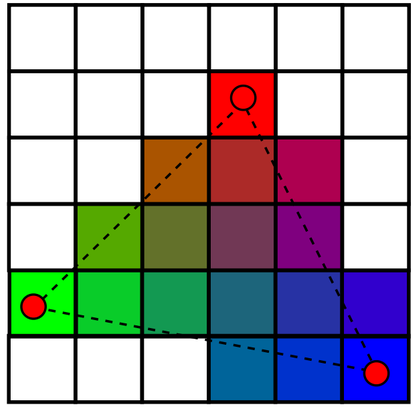


- *Throw rays from light-sources/camera*
- *Intersect rays with 3D shapes*
- *Pixel-wise computation*

+ 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*

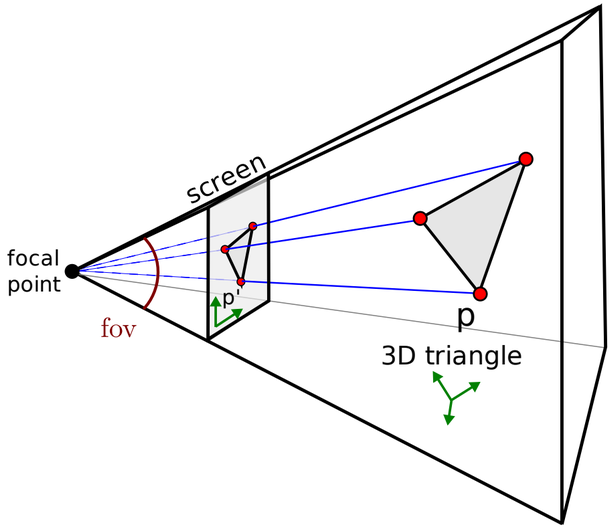
+ Efficiently implemented on GPU
- Limited to triangles
- No native effects (shadows, transparency, etc)

=> The standard real time rendering with GPU

Projection/Rasterization

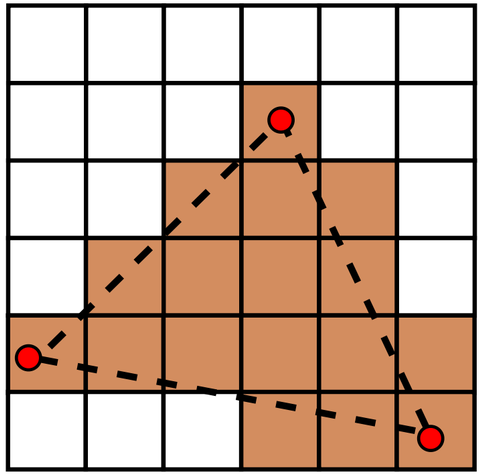
Object made of **triangles only**

1. **Project vertices of triangles**



- Projection computed as matrix operation (projective space $p' = \mathrm{M}\, p$)

2. **Rasterization**



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

=> At interactive frame rate ($\geq$ 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

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

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

Affine transform in 2D

General principle in the 2D case

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

Rotation $\mathrm{R} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$, Scaling $\mathrm{S} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$, Translation $(x + t_x, y + t_y)$ *(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).

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

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

Affine transform matrix

With the extra dimension (in 2D):

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

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

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

Similar in **3D** but with **4-components vectors**, and $\mathbf{4 \times 4}$ **matrices**.

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

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

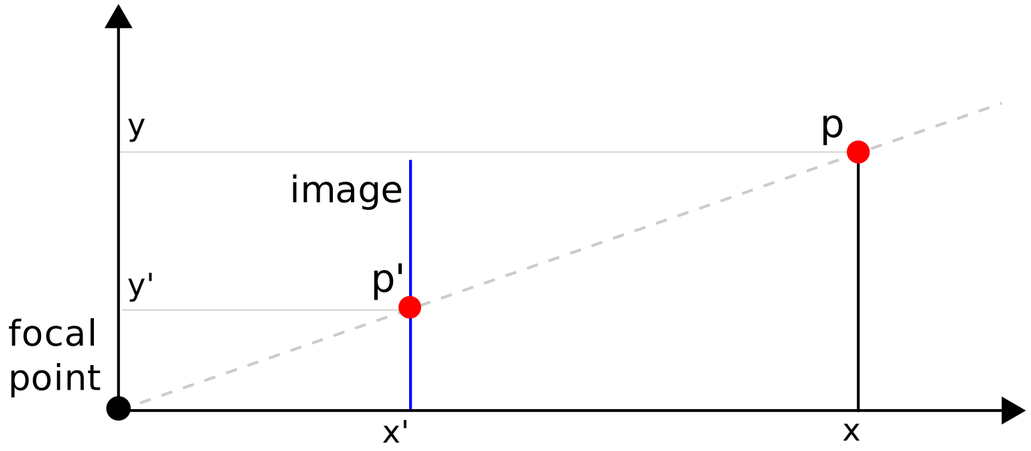
Note: vectors and points can be expressed

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

Perspective and projective space

Modeling perspective projection requires division.

ex. in 2D (1D projection)



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

Linear model using 3D vectors in projective space.

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

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

Projective space

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

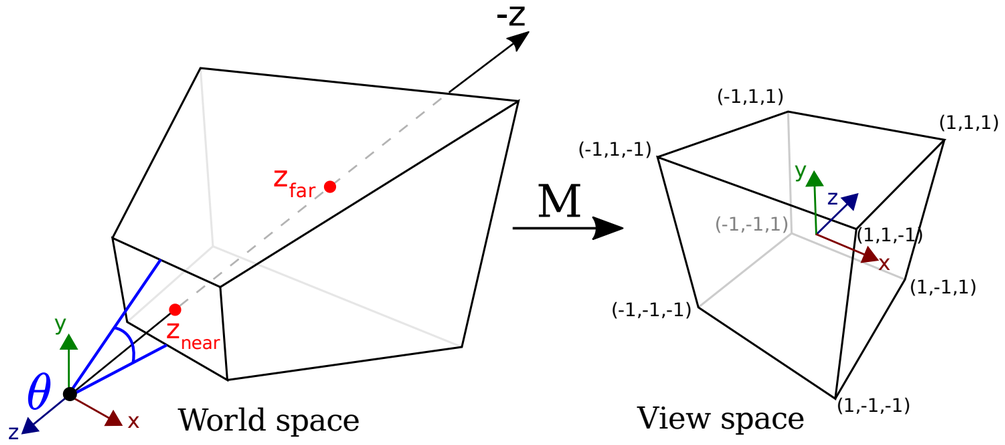
Real coordinates of points are obtained after normalization (division by z).

Perspective matrix

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
  - $p = (x, y, z, 1) \in$ frustum $\Rightarrow p' = (x', y', z', 1) \in [-1, 1]^3$.



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

In practice

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

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

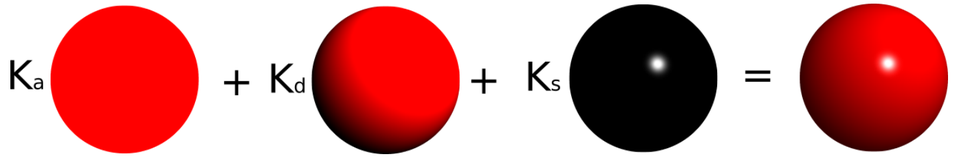
To which view space coordinates are mapped 3D world space points at z_{near}, z_{far} ?

Per-vertex normal and illumination

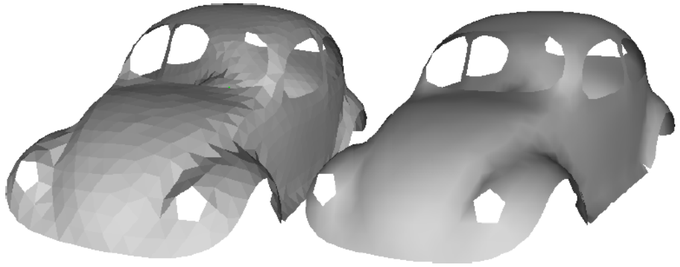
For smooth looking meshes, we define a **normal per-vertex**.

- Vertices are seen as samples on a smooth underlying surface

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

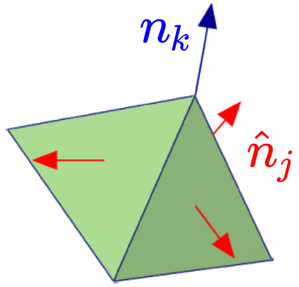


- *Phong shading* interpolates normals on each *fragments* of triangles, and compute illumination.



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

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

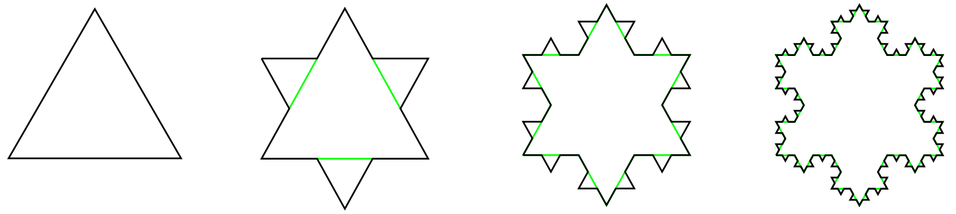


Fractals

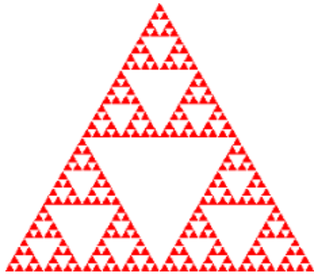
Idea: Recursively add self-similar details

Simple rule $\Rightarrow$ complex shapes

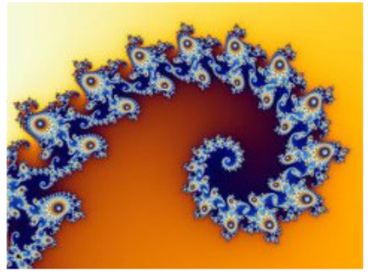
May look like complex natural details



Koch Snowflake



Sierpinski triangle, Shell:Oliva Porphyria



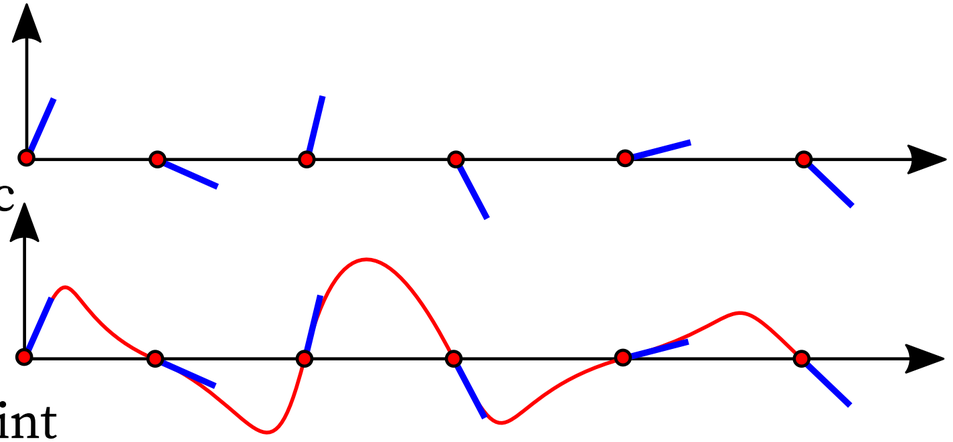
Perlin Noise

A widely used *noise* function.

Original article [An Image Synthesizer, Ken Perlin, SIGGRAPH 85]

Continuous Pseudo-random function

- Spatial variations are continuous, but non periodic
- Function can be evaluated at any point - deterministic



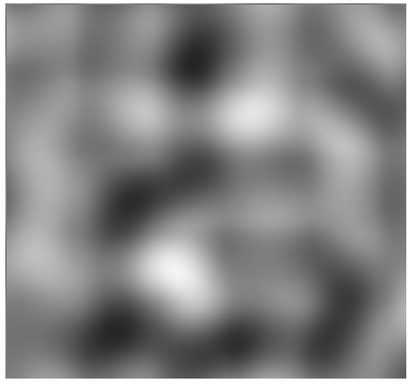
Creating a smooth function

- Compute pseudo-random gradient at each sample point

  Use some hash function

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

- Interpolate smooth cubic curve between each points



(Algorithm for nD: Simplex noise)

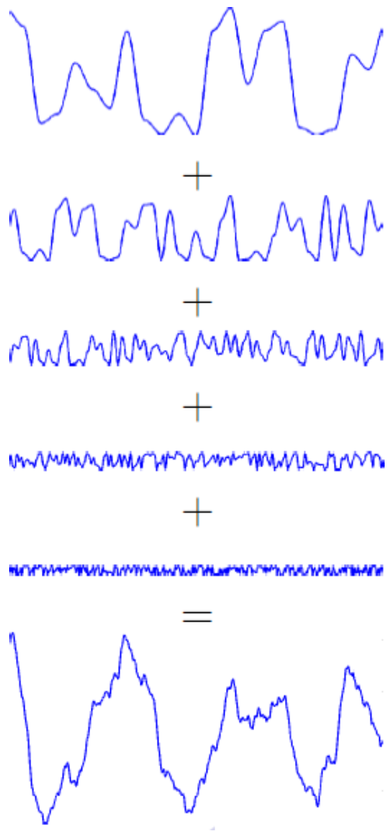
Fractal Perlin Noise

Sum over mutliple instance with increasing frequencies

f : Smooth Perlin Noise function

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

- N number of Octave
- α persistency ($1/\alpha$ attenuation)
- ω frequency gain



$f(x)$

$+$

$0.8 \ \ f(2\,x)$

$+$

$0.8^2 \ f(4\,x)$

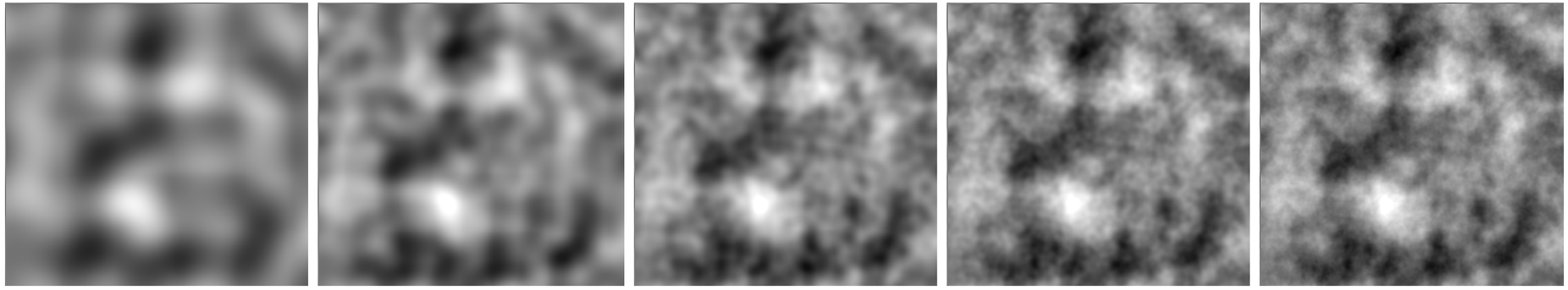
$+$

$0.8^3 \ f(8\,x)$

$+$

$0.8^4 \ f(16\,x)$

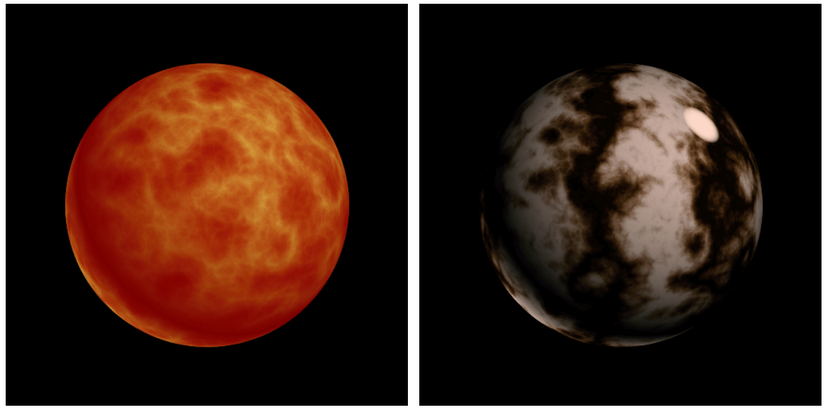
$=$



Perlin noise usage

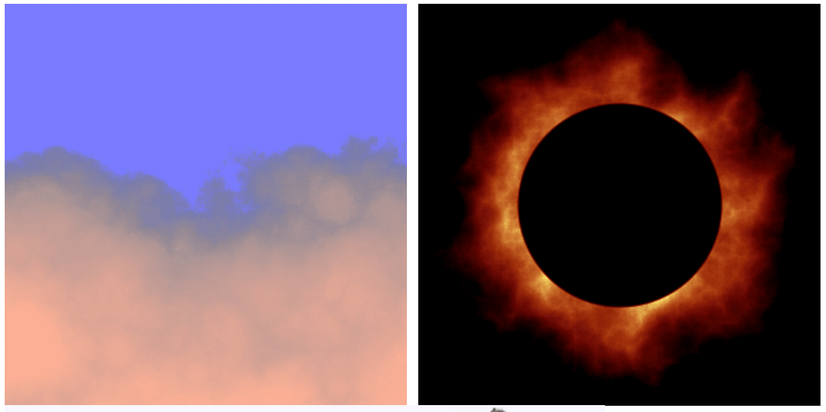
Material texture

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



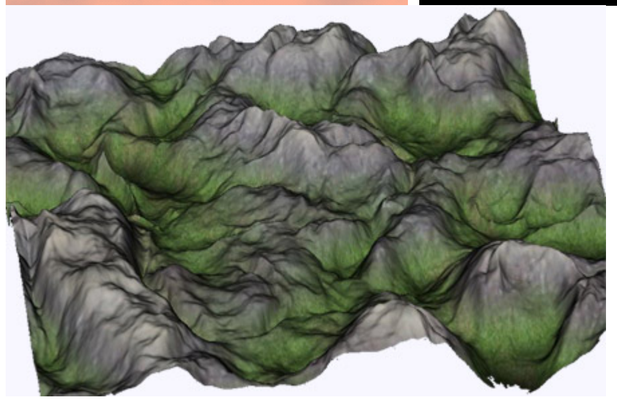
Animated textures

- Translation: $f(x, y + t)$
- Smooth evolution: $f(x, y, t)$



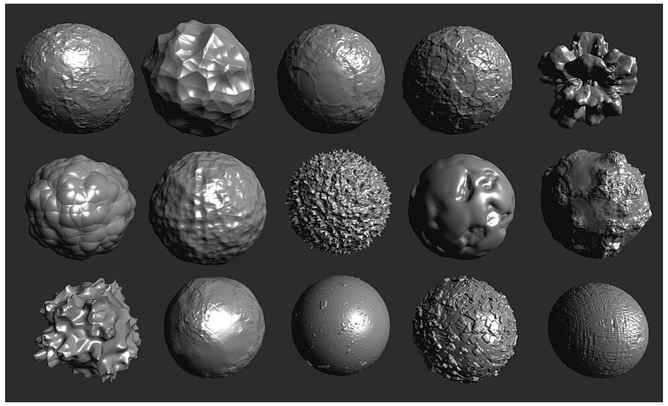
Mountain-looking terrain

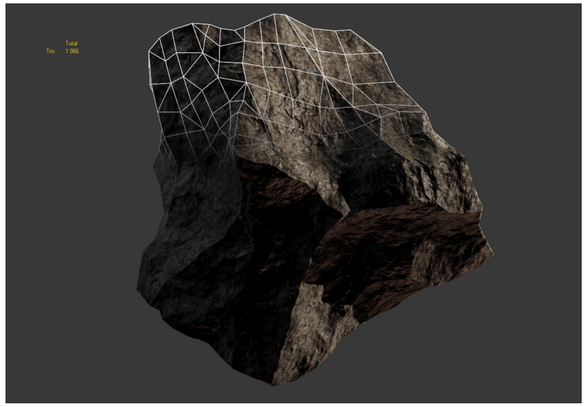
- $z = f(x, y)$

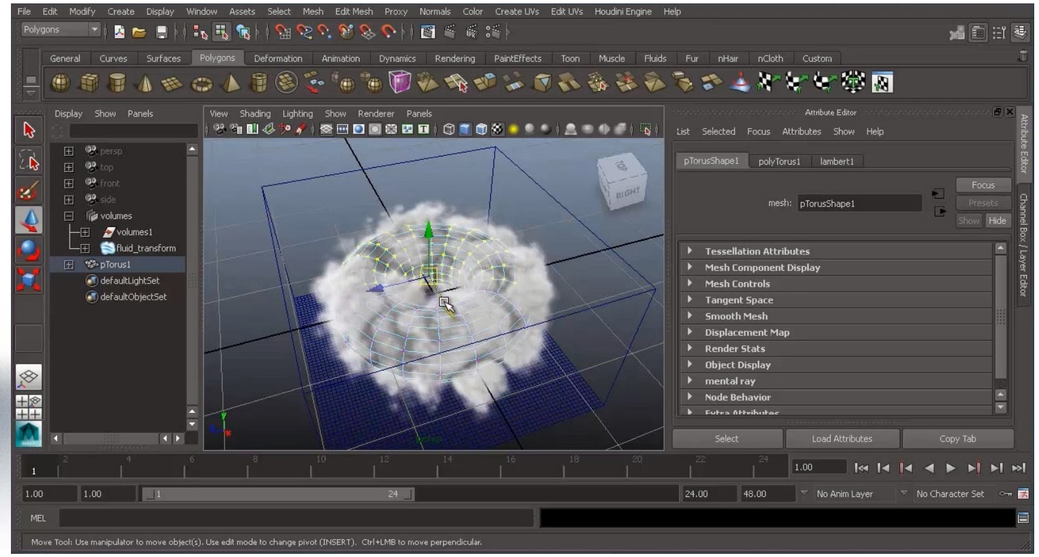


Perlin noise applications

In almost every complex shapes ...







Look at Shader Toy + Noise example

Geometry processing libraries

Mesh structure, algorithms, possibly viewers.

Development libraries

- LibIGL : Simple to use, efficient, geometry processing library
  Origin: ETH Zurich, Interactive Geometry Lab (IGL)
- CGAL : Advanced generic geometry processing library, Heavily templated
  Origin: Inria (Sophia)
- GeoGram : KISS geometry processing lib. Surface and volume structure.
  Origin: Loria (Nancy)

Viewer (+lib)

- Graphite : Surface and Volume geometry processing. API (Geogram) for geometry processing.
  Origin: Loria (Nancy)
- Meshlab : Mesh viewer and processing
  Origin: Visual Computing Lab, Pisa.

Software

- Blender : Full 3D modeler, animation, renderer (Open source)
  Provide python API for coding
  Real swiss knife for 3D Graphics !

Usefull CG programming library

Usefull libs

- Eigen : Efficient C++ Matrix toolbox
- GLM : GLSL compatible C++ structures (vec3, mat3, etc.)
- Assimp : Mesh loader
- DevIL : Image loader

Minimalistic GUI (OpenGL compatible)

- ImGui
- NanoGui
- AnTweakBar (ancestor of ImGui and NanoGui)

Full framework

- Qt : Cross plateform development including classes, GUI, visual designer, etc.

Warm up: Particle system trajectories

Particle system

Particle : element at a given position + extra parameters (mass, life time, etc)

Called **particles systems** in opposition to

- Rigid bodies - Solid objects with static shape
- Deformable bodies - Continuum material that can deforms

Pro

(+) Lightweight representation (both CPU+memory)

(+) Generic flexible model (spatial deformation, no connectivity, etc)

$\Rightarrow$ highly used in CG

Cons

(-) Simple model from physics point of view

Particles systems History

One of the first animated model in CG

Star Trek II

[Particle systems - A Technique for Modeling a Class of Fuzzy Objects, William T. Reeves, Lucasfilm, 1983] [pdf]

Karl Sims, Particle dreams, 1988 [link]

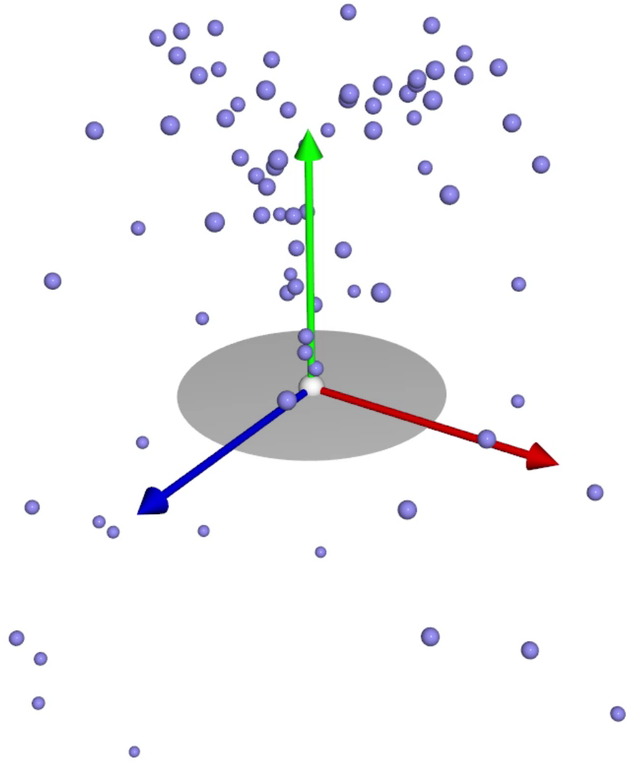
Example of Particle system

Free fall of spheres under gravity

- Geometrical representation of each particle: sphere
- Equation of motion $p(t) = \frac{1}{2}gt^2 + v_0 t + p_0$

- Initial position and speed may be placed at random positior
- Each particle may have a different life time

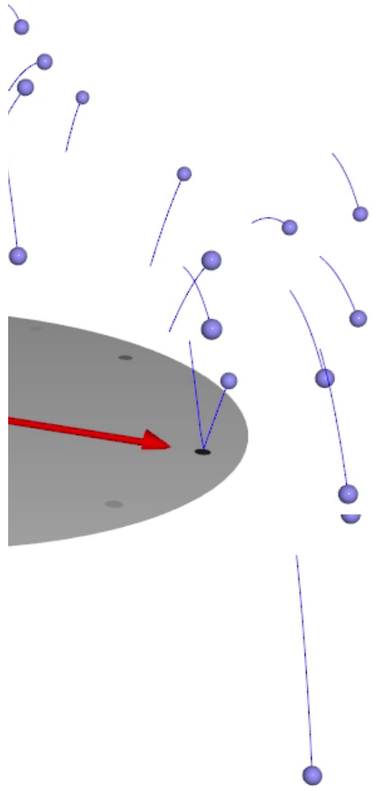
- *What are the parameters used for p_0 and v_0 in this example ?*



Bouncing spheres

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

  - *Consider a particle emited at time* $t = 0$
  - *At what time* t_i*, the particle touch the floor ?*
  - *What is the new speed after impact ?*
  - *What is the complete equation of trajectory ?*

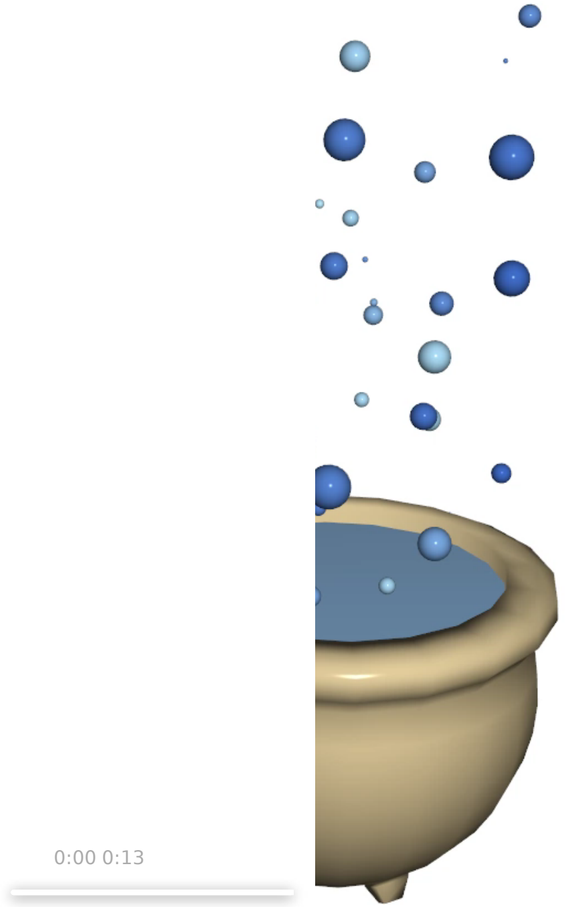


0:00 / 0:08

General motions

Motion equation is not restricted to physics-based equations

- *What are the parameters associated to each particle ?*
- *What are the corresponding equations of motion ?*



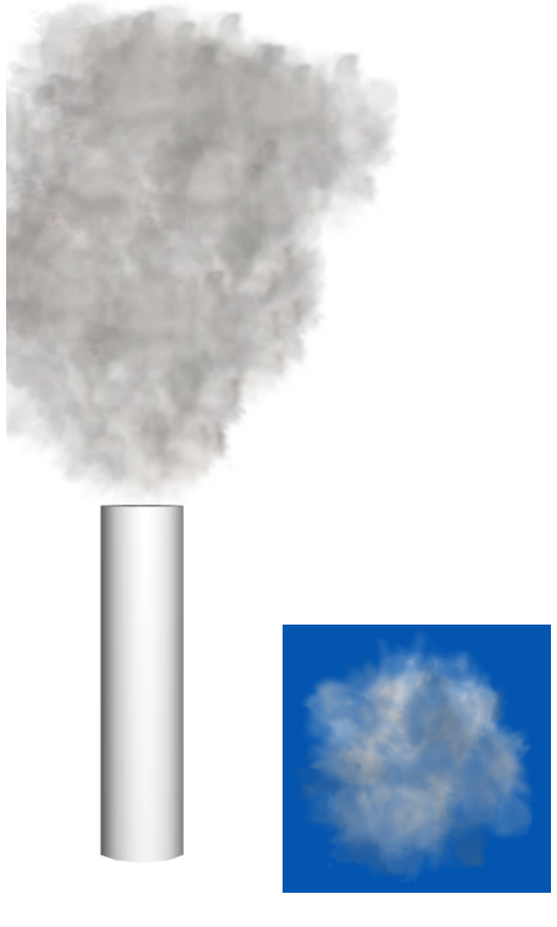
Billboards, Impostors, Sprites

Particles can be displayed as small images/thumbnails

In practice

- Each particle is displayed as a quadrangle
- A texture is mapped on the quad

- The texture can contains transparency : quad geometry is invisible
- 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)

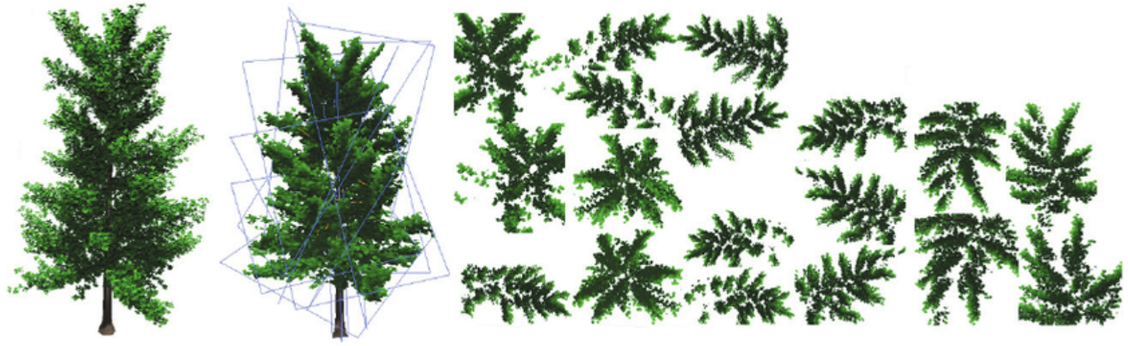


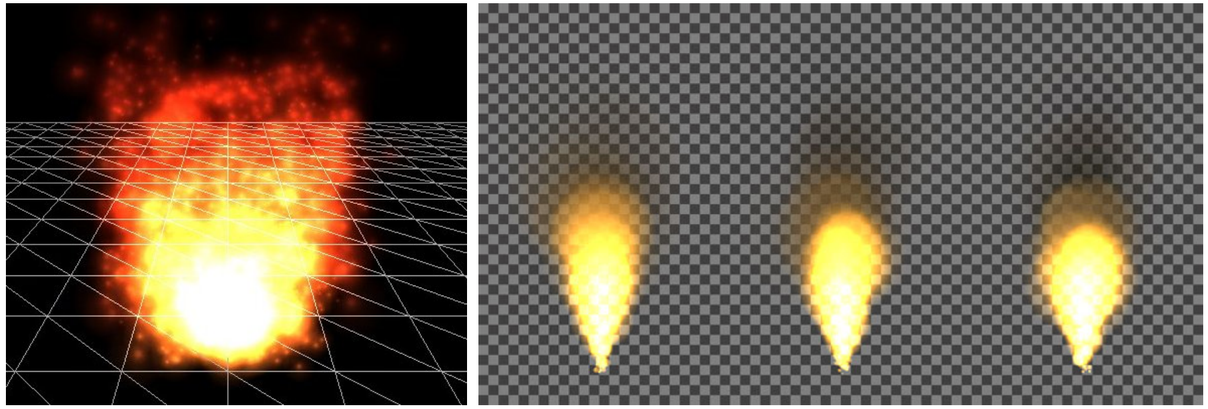
0:00 / 0:16

Usage of billboards

Large use of billboards for complex models

Vegetation, fire, smoke, etc.





Billboards, Impostors, Sprites

Follow up of the example

How to model the following animation ?

Use case in production

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

The Lord of the Rings

Use case in production

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

The Lord of the Rings

Use case in production

Full Making-Of

The Lord of the Rings

Introduction to Computer Animation

Animation in production

Animation in Computer Graphics

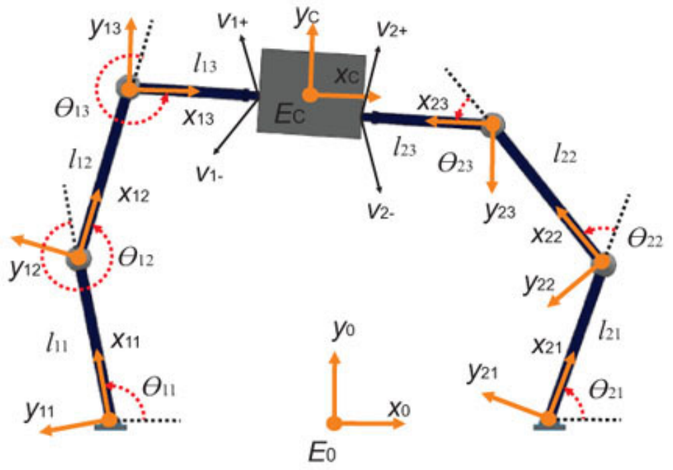
Two main ways to describe animation

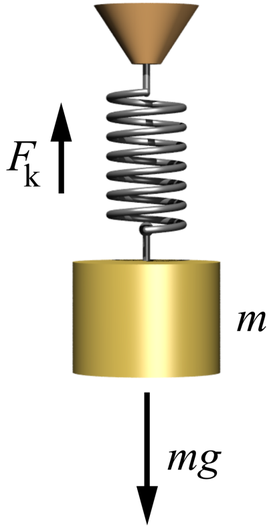
1- Kinematics

Shape deformation/motion without knowledge of the physical cause of the motion

2- Dynamics

Motion through forces acting on shapes (ex. physically based simulation)





Animation in Computer Graphics

Ways to generate animation

1- Descriptive animation

Fully designed trajectory

2- Motion tracking

Real data aquisition

3- Procedural generation

Automatic synthesis

4- Physically based simulation

5- Learning-based synthesis

Descriptive Animation

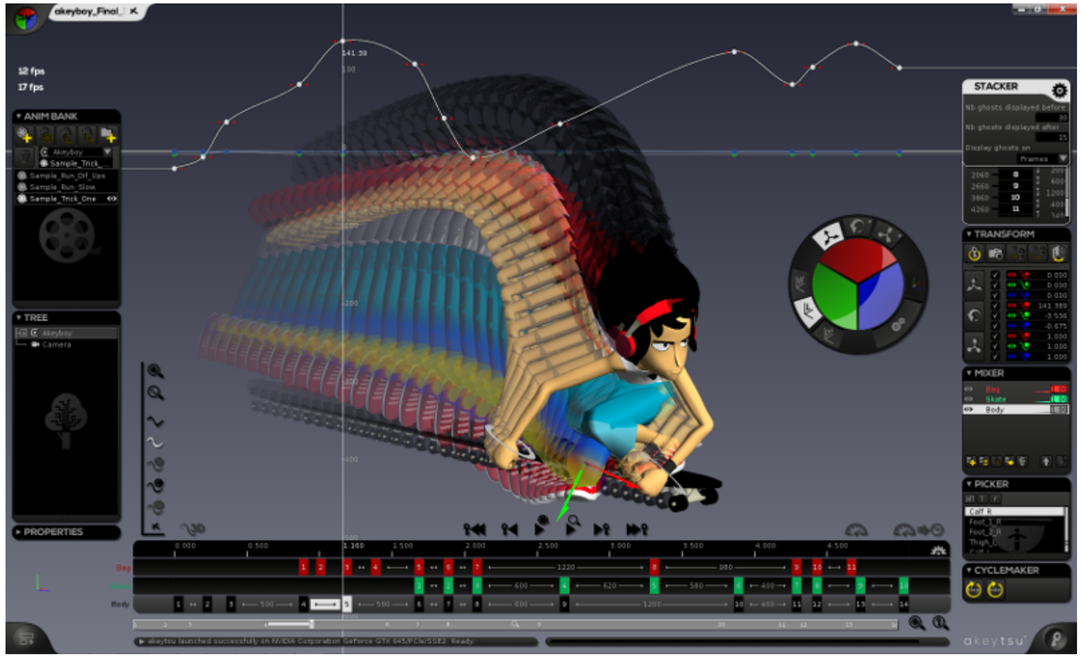
The artist/an algorithm fully describes the motion and deformation.

pro

+ Full control on the result

cons

- May introduce un-physical artifacts



History of CG Animation

First production of animated films (Disney)

- 30 drawings / sec

Principle

- *Animator in chief*: Create **Key Frames**
- *Assistants*: Fill the *in between* (secondary drawings)

=> Principle of **Key Framing**

(c) Walt Disney Company, from "The Illusion of Life"

Key Framing in CG

- Create *manually* a set of **Key Frames** at specific time steps
- Compute automatically intermediate frames (30 fps) using **interpolation**

Production Pipeline

Production Pipeline

Fundamental structure for VFX/animation movie production.

Pre-production

Story, script, concept art

Storyboard

Animatic

Production

Layout

Special Effects

Modeling

Rigging

Shading

Texturing

Lighting

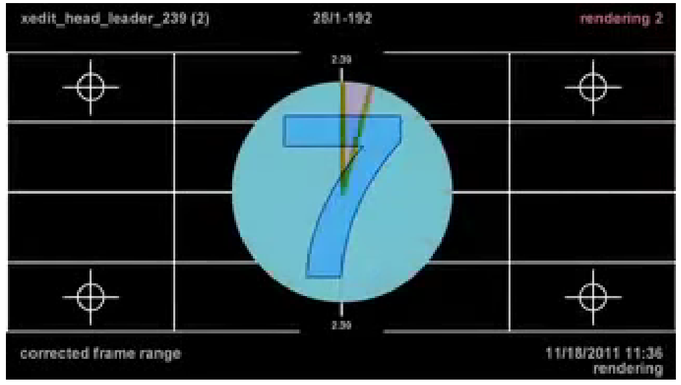
Animation

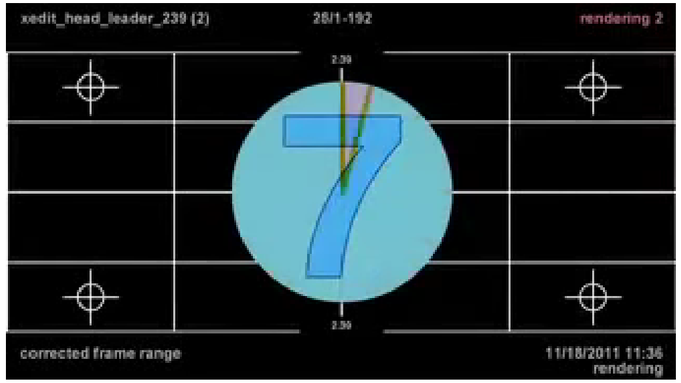
Rendering

Post-Production

Compositing

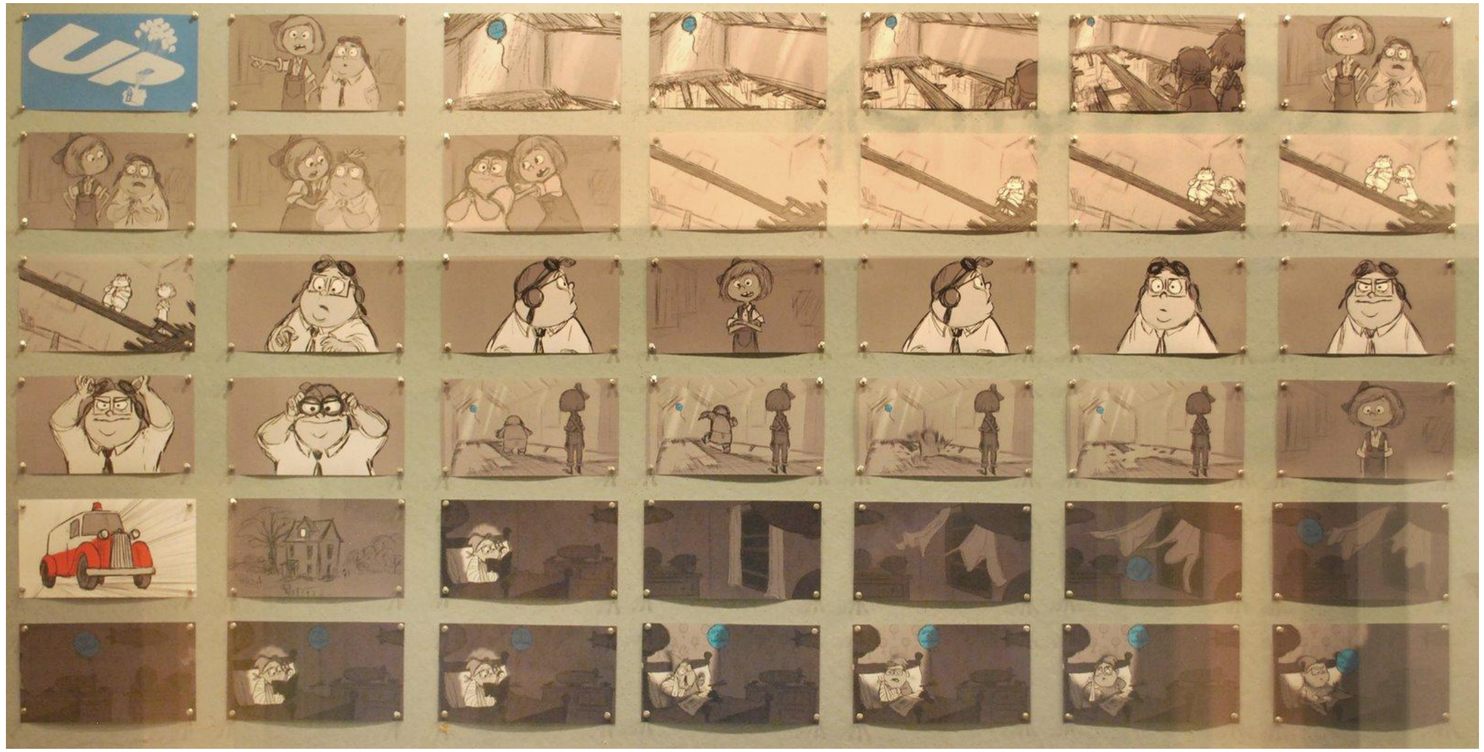
Editing




xedit_head_leader_239 (2)
28/1-192
rendering 2
239
239
corrected frame range
11/18/2011 11:36
rendering

0:00 1:15

Story-board



Few 2D drawings: express the story

Before mostly non-technical - artistic/creative based
Curently Strong increase of Storytelling-related researchs in CG

ex. link:https://studios.disneyresearch.com/[Disney Research Studios], link:https://team.inria.fr/anima/[Anima team at Inria]

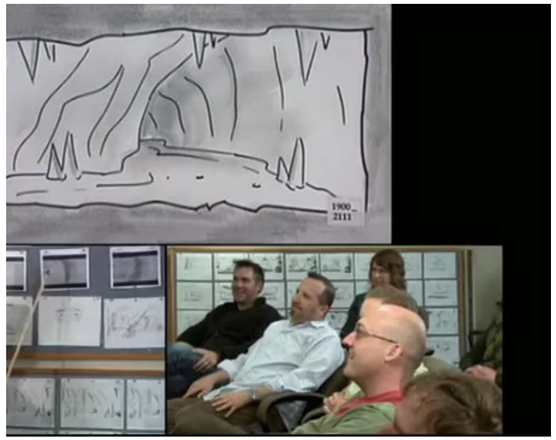
Animatic

$\simeq$ Animated story-board, Various format

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

0:00 / 3:36

0:00 / 2:58



0:00 0:53

Layout: Moving into 3D

1st step: Rough 3D modeling and placement of

- Camera: visual field, perspective

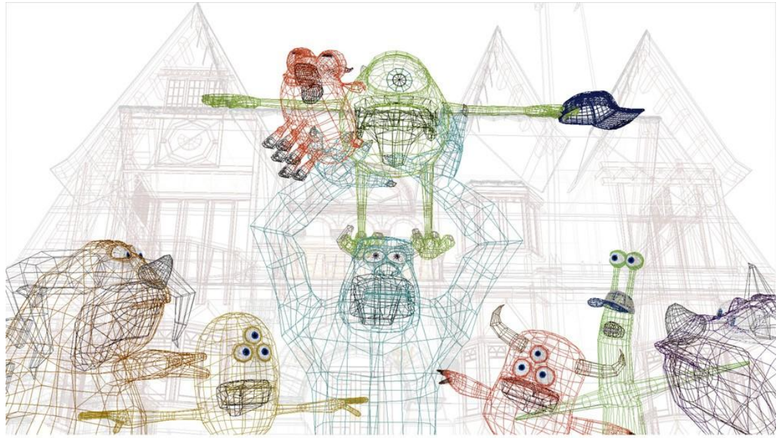
  3D much more constrained than 2D drawings

- 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

In production

- Polygonal mesh modeling

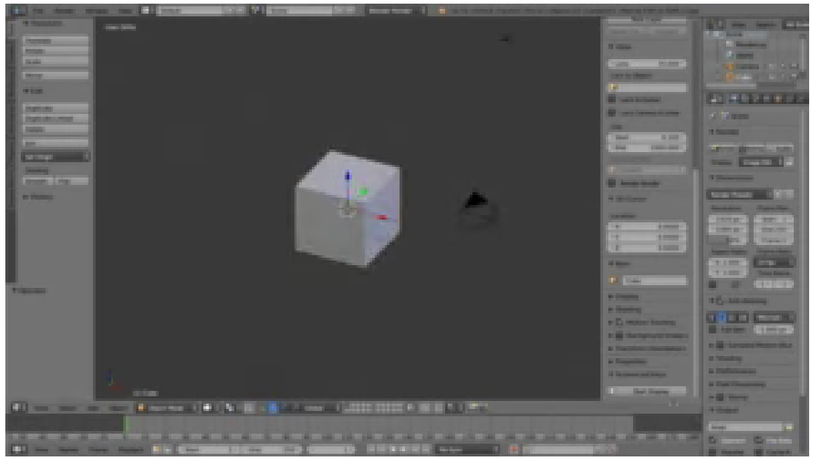
  Coarse to fine approach

  1. Low res modeling (extrusion)

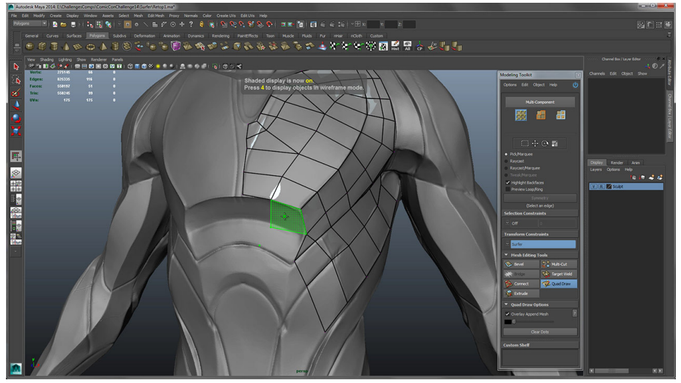
  2. Subdivide, Refine

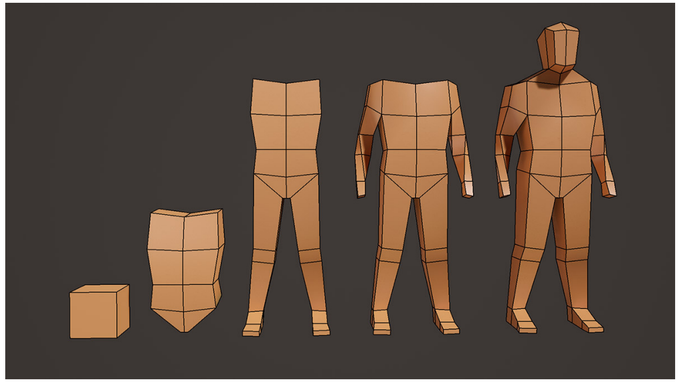
- Parametric (NURBS) modeling

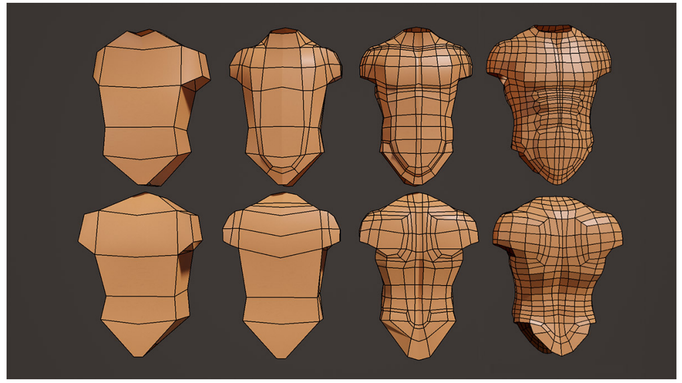
Everything else is "VFX"



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







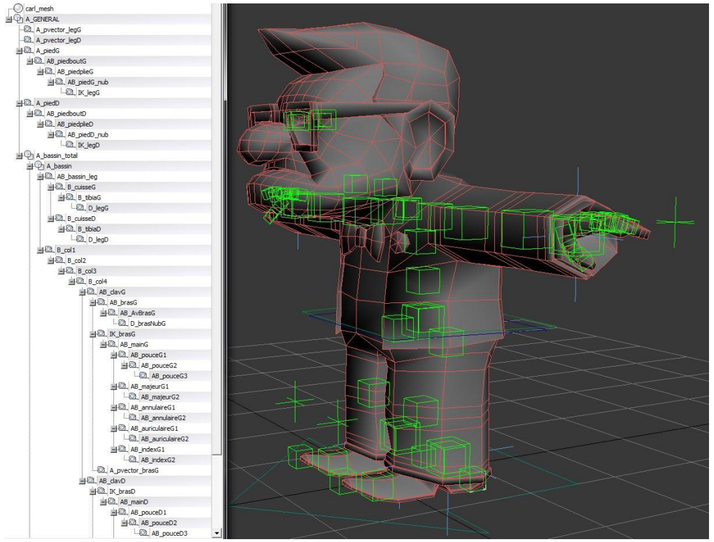
Rigging

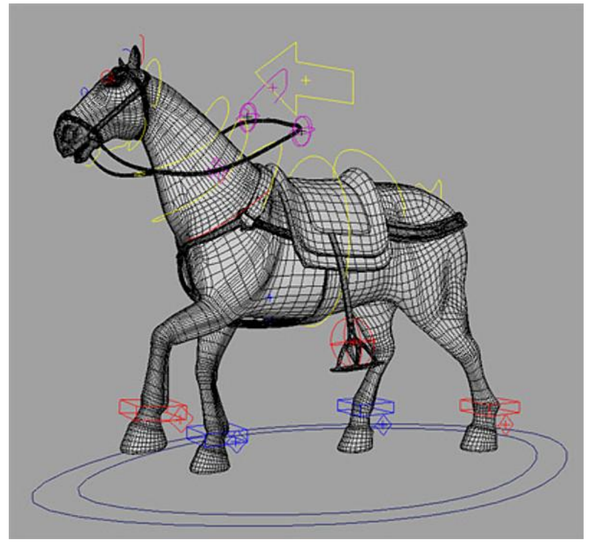
Attach deformation handles to the mesh

Each handle (controler) is associated to a deformation - degrees of freedom

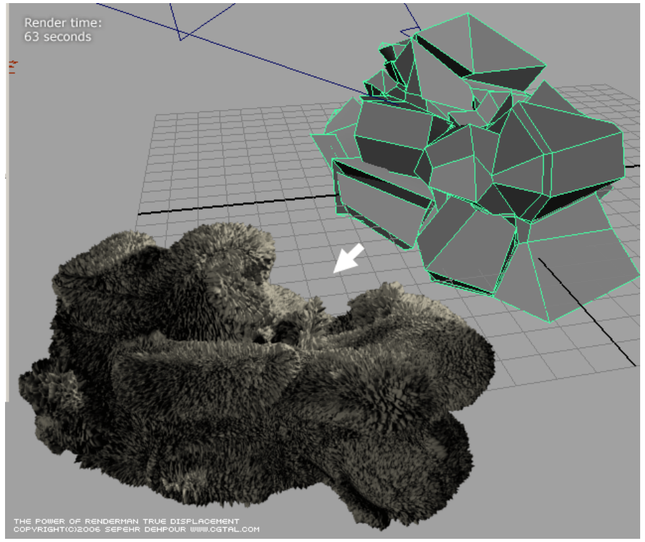
Rigging is a technical part

Python script, Mel (Maya), Lua, etc.

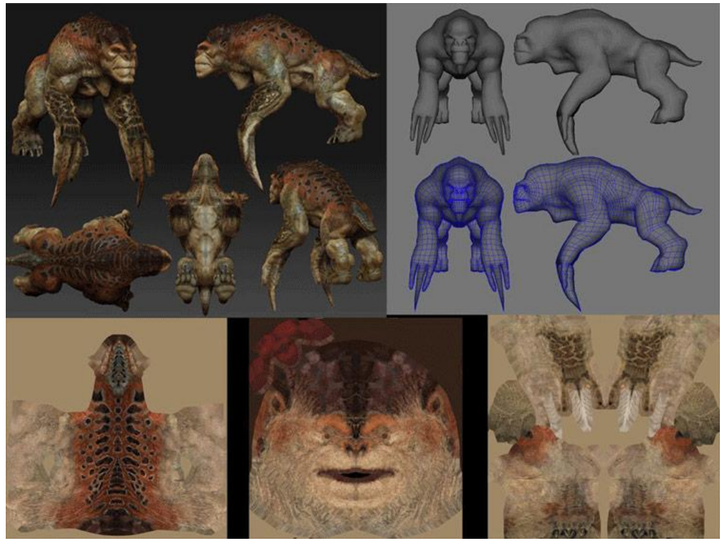




Modeling appearance - Rendering purpose

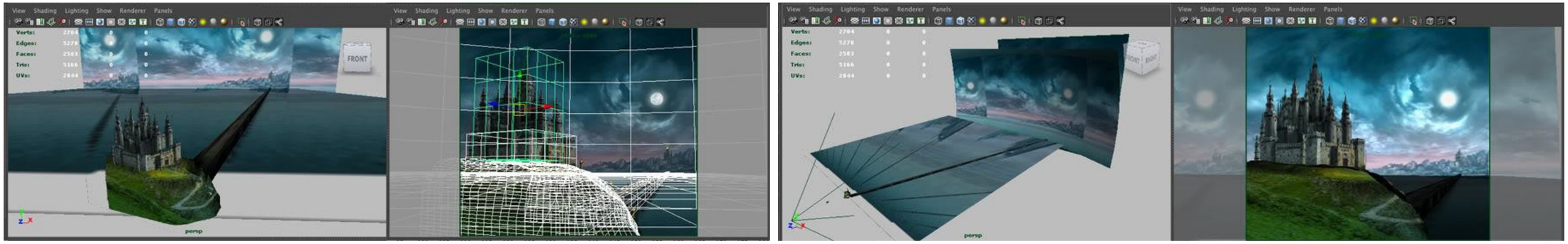


Shading



Texturing

Lighting



Mate painting

Animation

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)

0:00 / 0:40

Result

Up to 75% of artists production studios are animators

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

One animator $\longrightarrow$ 1-10 s of animation per day

Animation sub-parts

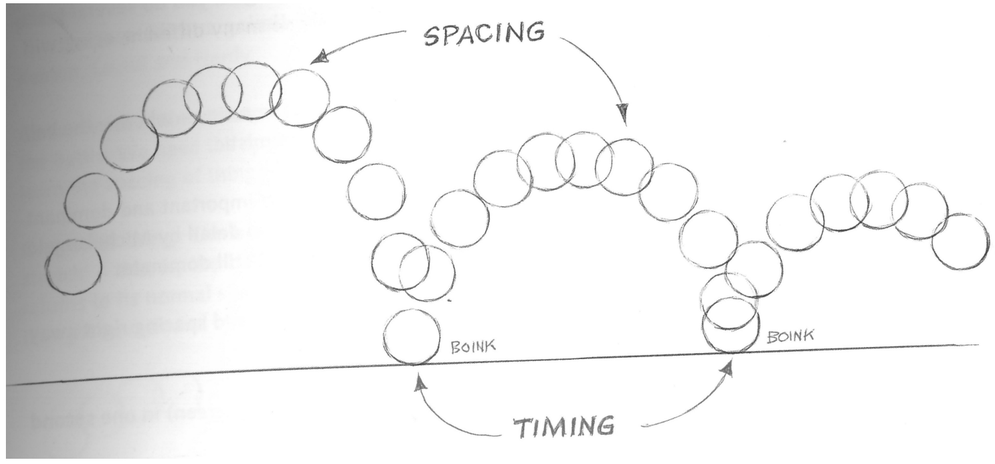
1- **Posing** the *key frames*

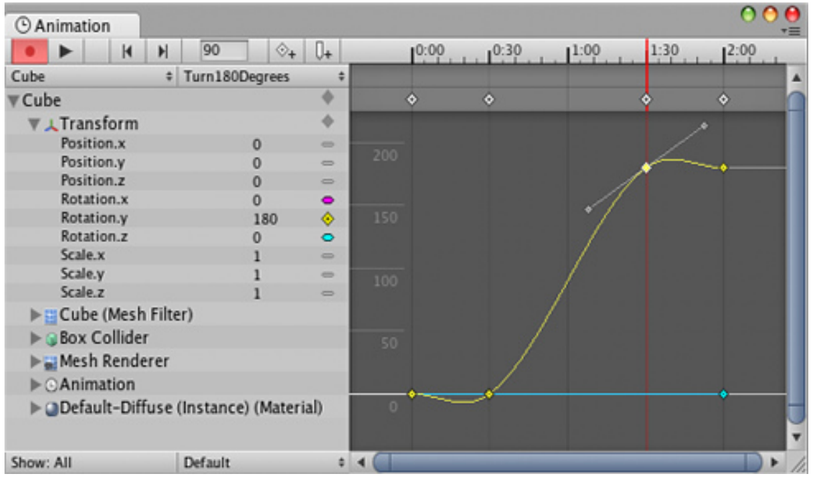
Set the main general posture of the character

*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)

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

Compositing

Blend all layers: Rendered and real ones

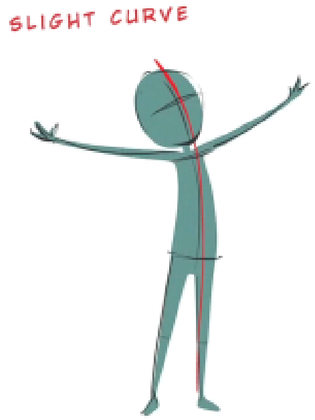
Note: Rendering of color layers but also depth and normals.

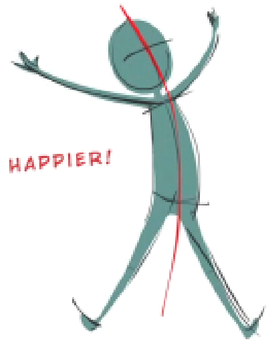
Main software: Nuke (Foundry)

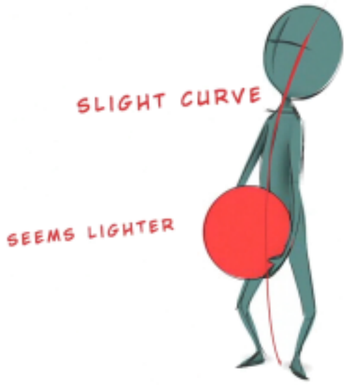
Expressive animation

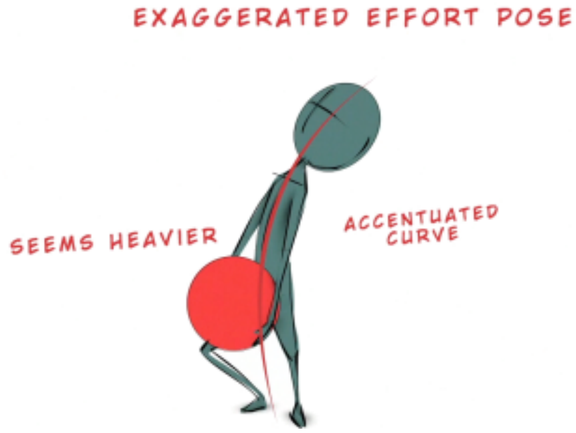
Character Animation Posing - Line of Action

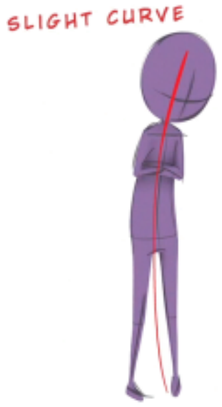
- Line of action: *Medial axis* expressing the character pose
- Express *statically* the dynamic of the action
  - Unstable pose ⇒ Dynamic action/motion

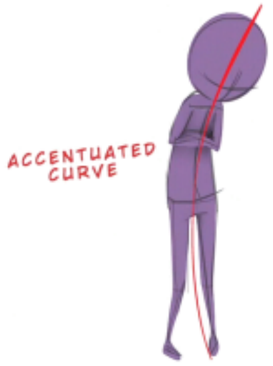












Principles of animation

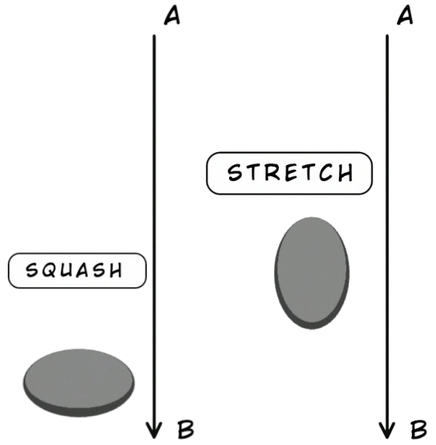
- Interpolation between realistic poses isn't enough for expressive animation
- *12 principles of animation* by Disney *Illusion of Life, 1981*

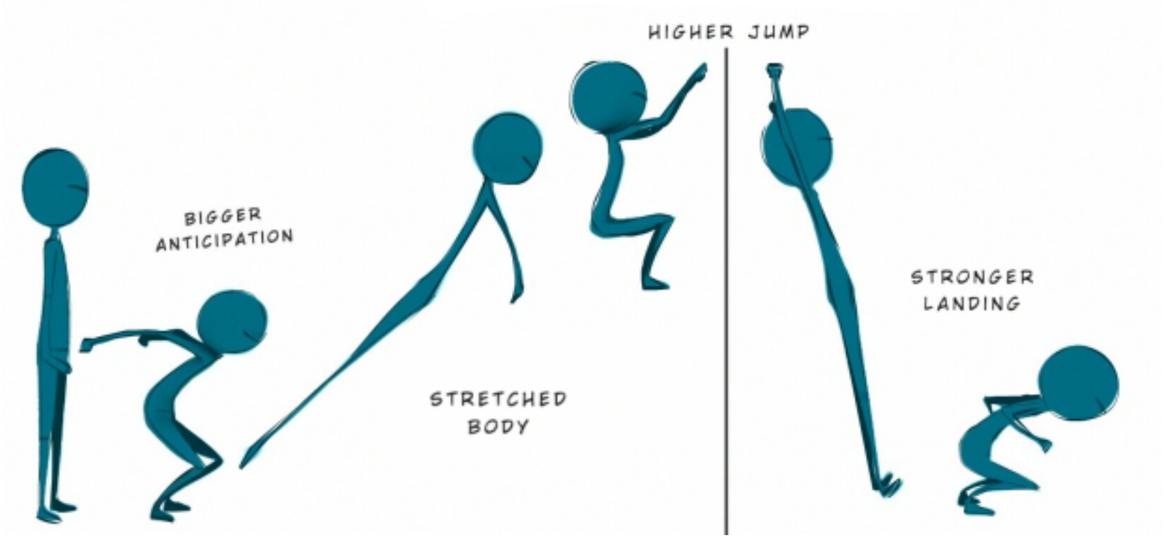
1. Timing
2. Spacing
3. Slow-in, Slow-out
4. Squash & Stretch
5. Anticipation
6. Follow Through
7. Secondary Action
8. Exaggeration
9. Appeal
10. Arcs
11. Staging
12. Straigh Ahead/Pose to Pose

Expressive animation

Squash & Stretch

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

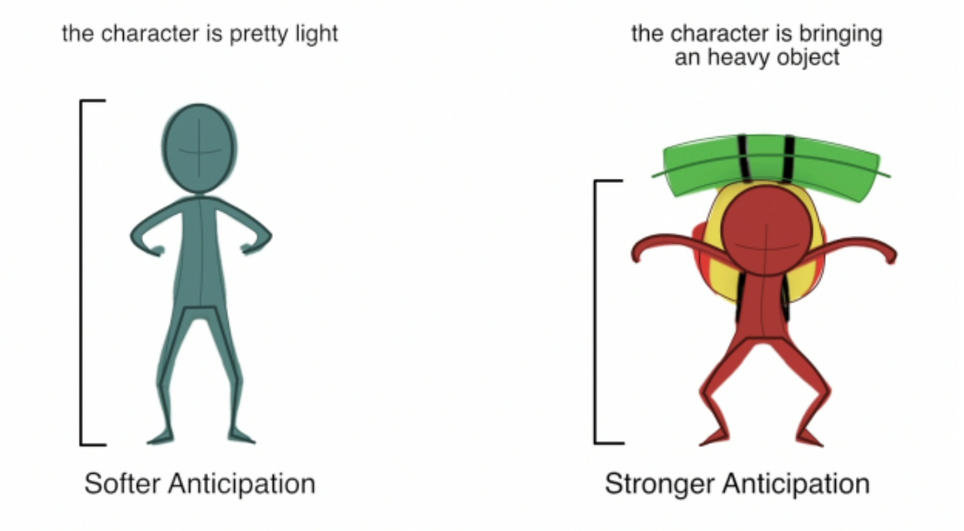




(C) Chiara Porri

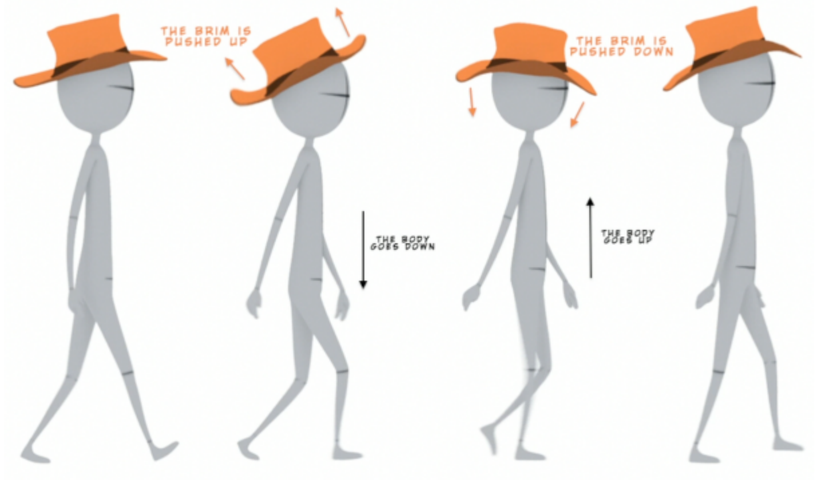
Expressive animation

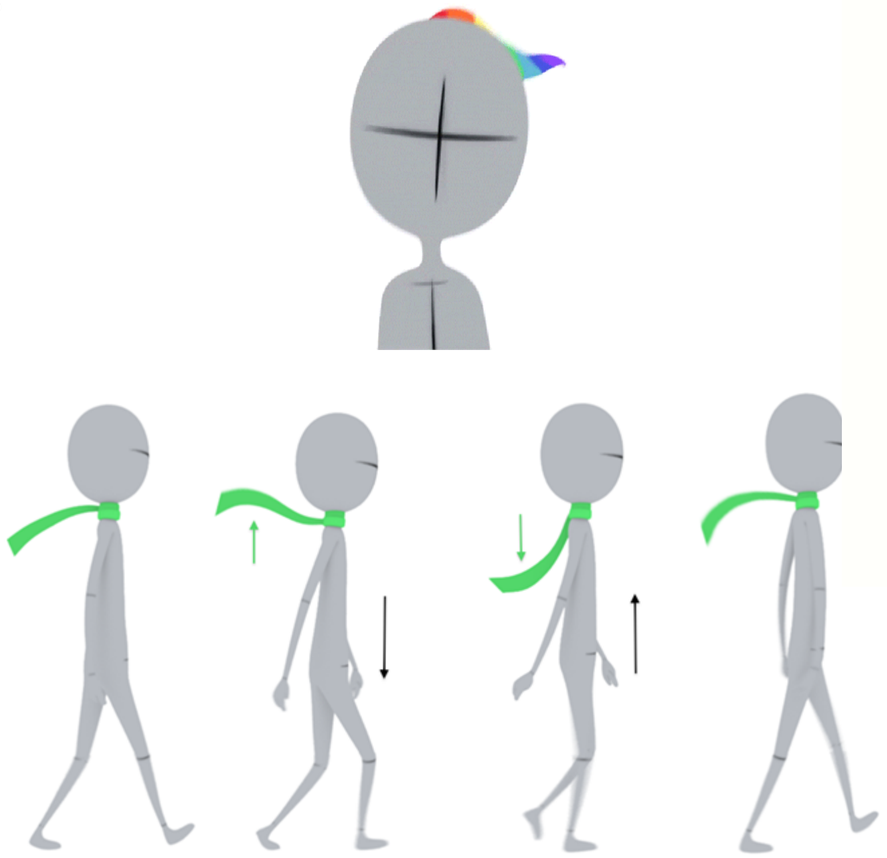
Anticipation

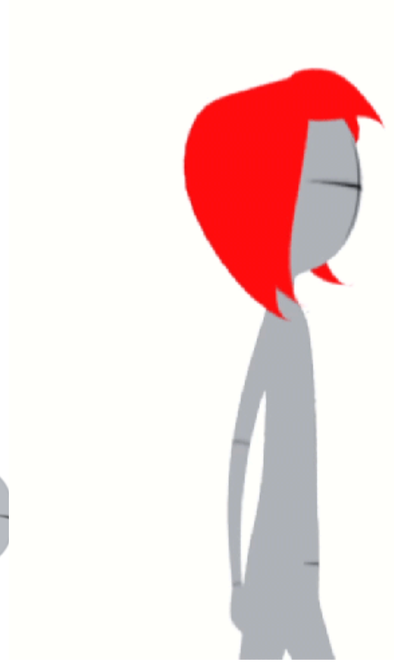


Expressive animation

Follow Through / Secondary motions





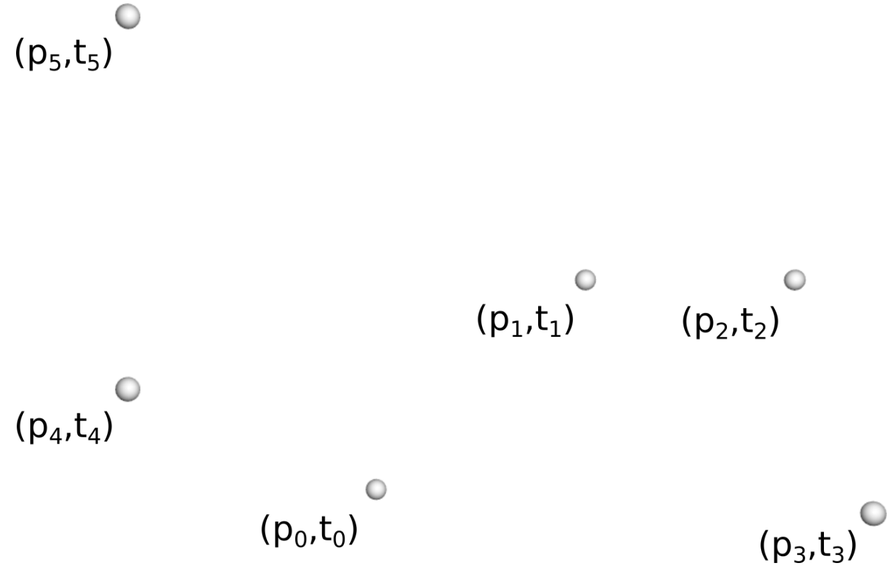


(C) Chiara Porri

Interpolate positions

Objective

- Given a set of key-positions (positions+time) we want to find an interpolating space-time curve

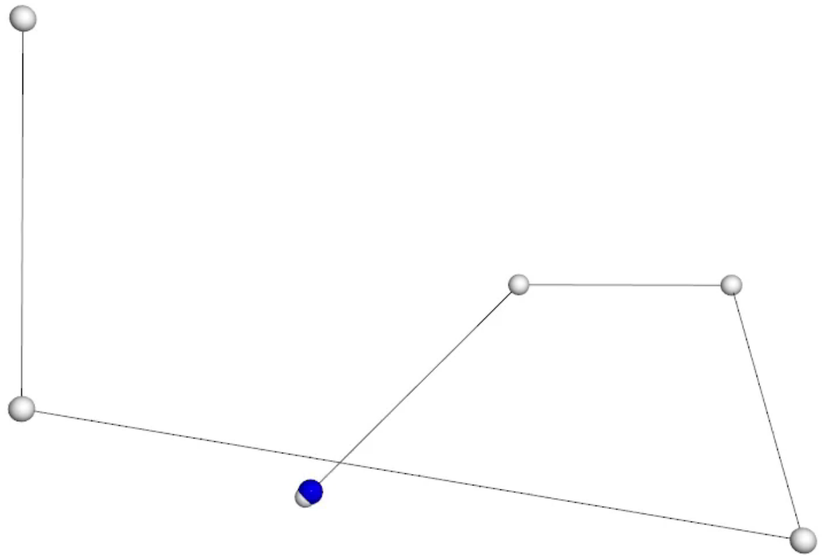


- **Input** $(p_i, t_i), i \in [0, N-1]$
- **Output**
  - Space time curve $p(t), t \in [t_0, t_{N-1}]$
  - Space time curve $p(t_i) = p_i$

Linear Interpolation

- Simplest solution: linear interpolation between each sample pairs

- $\forall t \in [t_i, t_{i+1}]$, $\begin{cases} p(t) = (1 - \alpha(t))\, p_i + \alpha(t)\, p_{i+1} \\ \alpha(t) = \dfrac{t - t_i}{t_{i+1} - t_i} \end{cases}$



Pros

- Simple
- Constant speed between keyframes

  Easy to ajust

Cons

- Non smooth trajectory
- Generates straight segments

  Artists prefer non straight trajectories for "real" looking motion

Smooth curve

Objective: Generate a **smooth** interpolating space-time curve

Classical Idea Use polynomials curves

$$\begin{cases} p(t) = \sum_{i=0}^{N-1} \alpha_i(t)\, p_i \\ \alpha_i(t) = \sum_{j=0}^{d} c_i^j\, t^i \end{cases}$$

(α_i) polynomial basis function of degree d

Which polynomials/degree choose ?

Lagrange polynomial interpolation

Naive idea: Interpolate all points at once

$$\forall t \in [t_0, t_{N-1}] \ , \quad p(t) = \sum_{i=0}^{N-1} \alpha_i(t) \ p_i \qquad \forall i \in [0, N-1] \ \ p(t_i) = p_i$$

Degree of polynomial : $N-1$

- Known solution: **Lagrange polynomial**

$$p(t) = \sum_{i=0}^{N-1} \alpha_i(t) \ p_i \qquad \alpha_i(t) = \prod_{k=0, k \neq i}^{N-1} \frac{t - t_k}{t_i - t_k}$$

- Explanation: By construction $\alpha_i(t_i) = 1$ and $\alpha_i(t_k) = 0$

(+) Interpolate all points

(-) Large oscillations between samples for large degree.

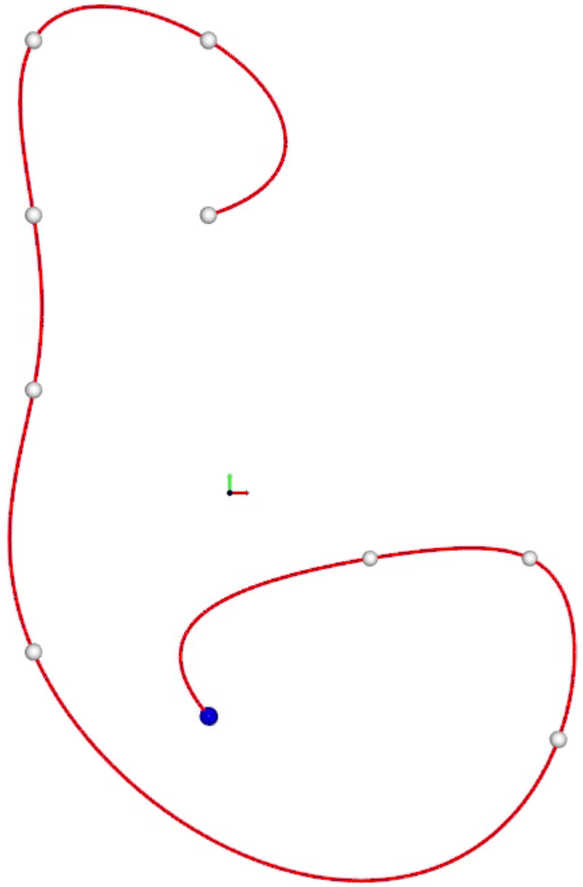
(-) Non local influence

=> Not used in practice

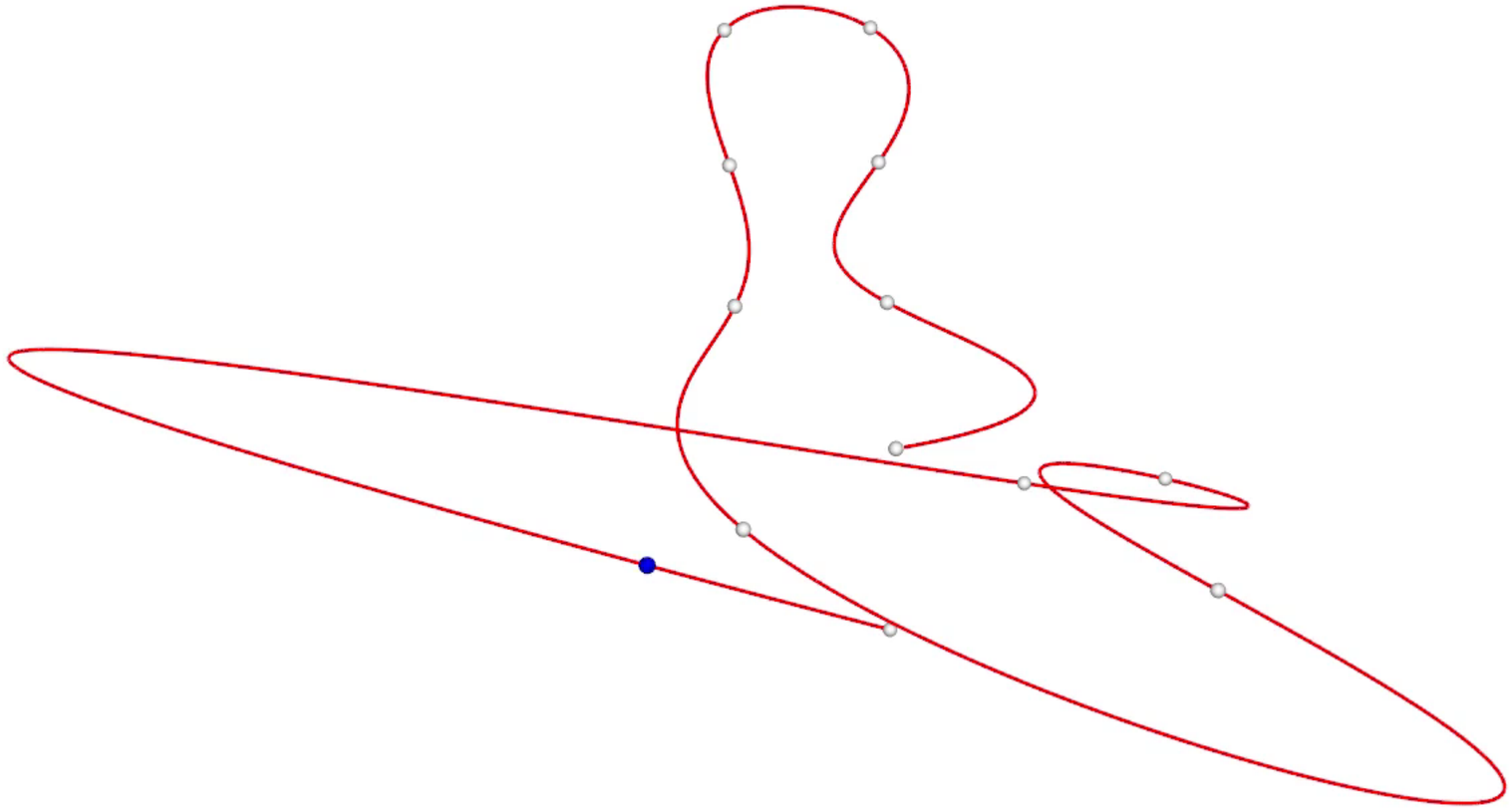
Lagrange polynomial interpolation : Comparison

Ex. Global effect on curve when two samples are added.

10 samples



12 samples



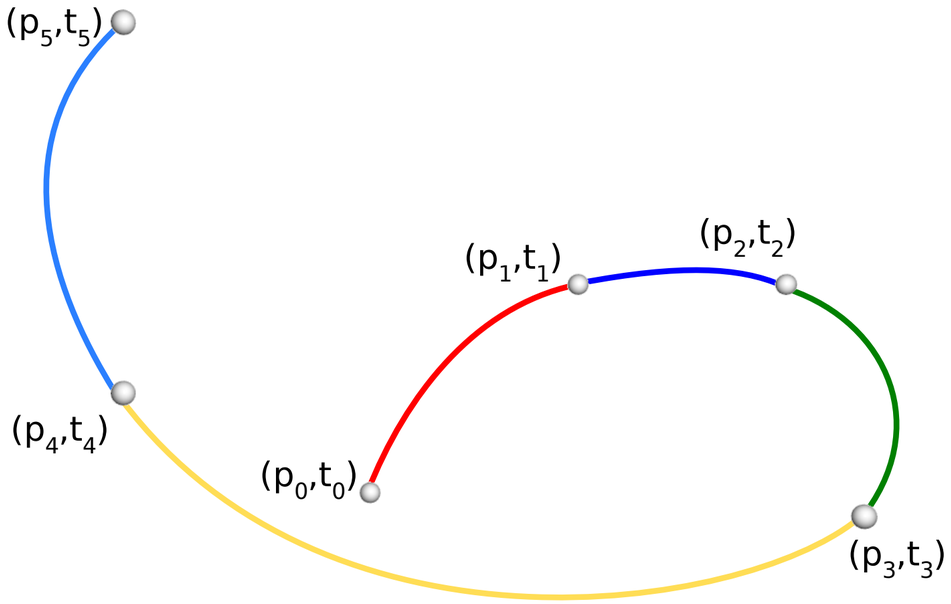
Spline

Idea

- Define on each part a polynomial
- Smooth junctions between them.

How to choose the polynomial

- Sufficiently high degree to be smooth
- Sufficiently low degree to avoid oscillations



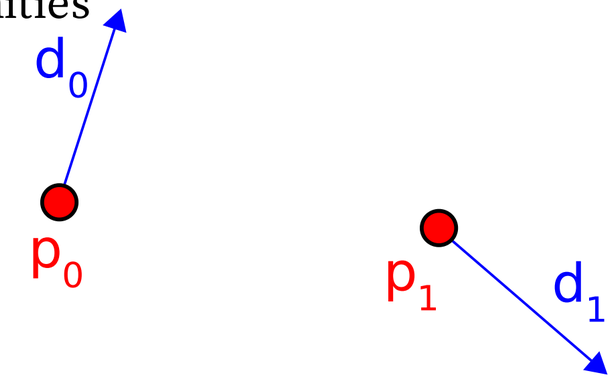
=> In Graphics cubic polynomials are often used

Allows up to C^2 junctions

Each color is another polynomial

Hermite interpolation

Hermite interpolation : cubic curve interpolating points and derivatives at extremities

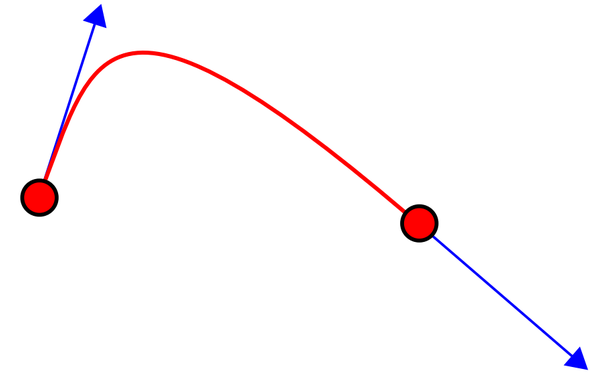


Consider the following constraints

$$\begin{cases} p(s) = c_3 s^3 + c_2 s^2 + c_1 s + c_0 \\ p(0) = p_0,\ p(1) = p_1,\ p'(0) = d_0,\ p'(1) = d_1 \end{cases}$$

$\Rightarrow$ System of equations

$$\begin{cases} & & & c_0 & = p_0 \\ c_3 + & c_2 + & c_1 + & c_0 & = p_1 \\ & & c_1 & & = d_0 \\ 3\,c_3 + & 2\,c_2 + & c_1 & & = d_1 \end{cases} \Rightarrow \begin{cases} c_0 = p_0 \\ c_1 = d_0 \\ c_2 = -3p_0 + 3p_1 - 2d_0 - d_1 \\ c_3 = 2p_0 - 2p_1 + d_0 + d_1 \end{cases}$$



$$\forall s \in [0,1]\ ,\ \ p(s) = (2s^3 - 3s^2 + 1)\,p_0 + (s^3 - 2s^2 + s)\,d_0 + (-2s^3 + 3s^2)\,p_1 + (s^3 - s^2)\,d_1$$

For arbitrary $t \in [t_i, t_{i+1}]$, we set $s = \dfrac{t - t_i}{t_{i+1} - t_i}$

Interpolating curve

Our initial problem: set of multiple keyframes position+time

Two solutions

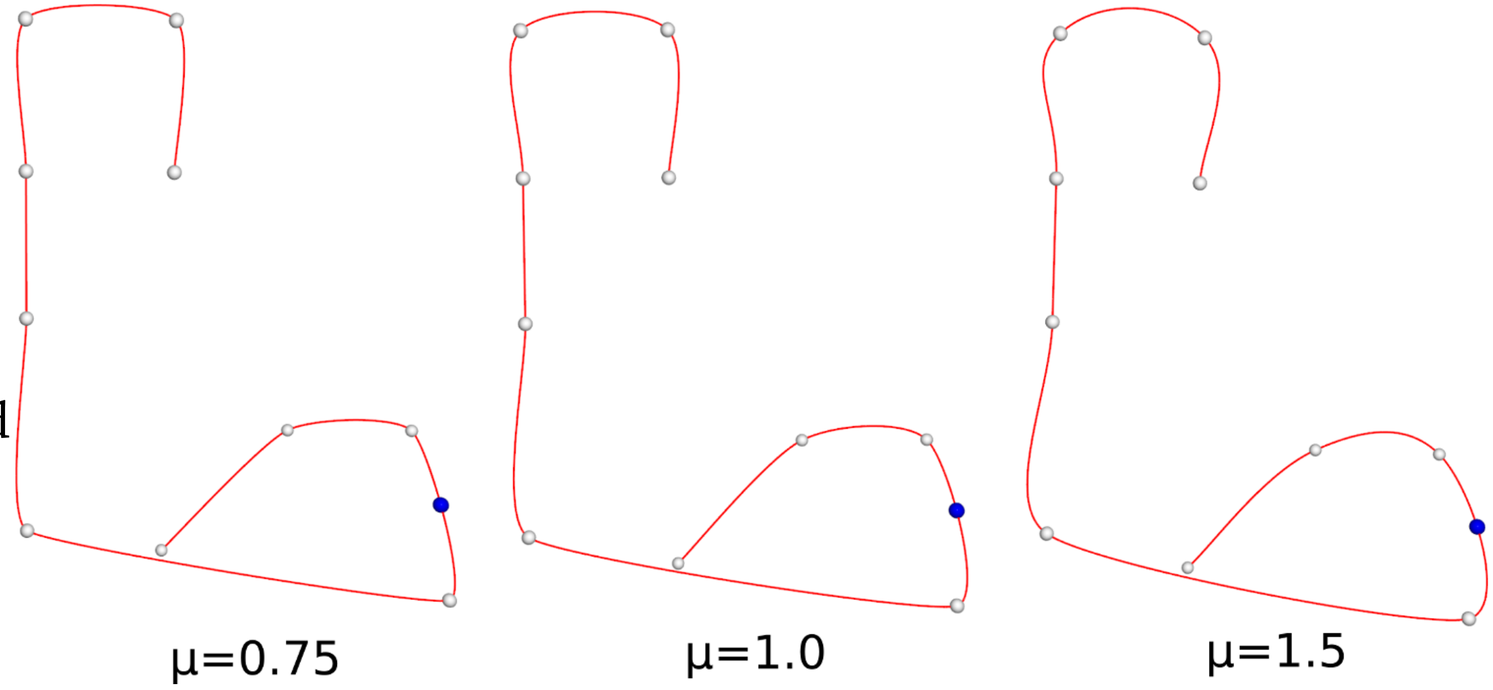
- Set explicitely derivatives for each keyframe - *teadious*
- Compute automatically plausible derivatives from surrounding samples - *often used*

Cardinal spline

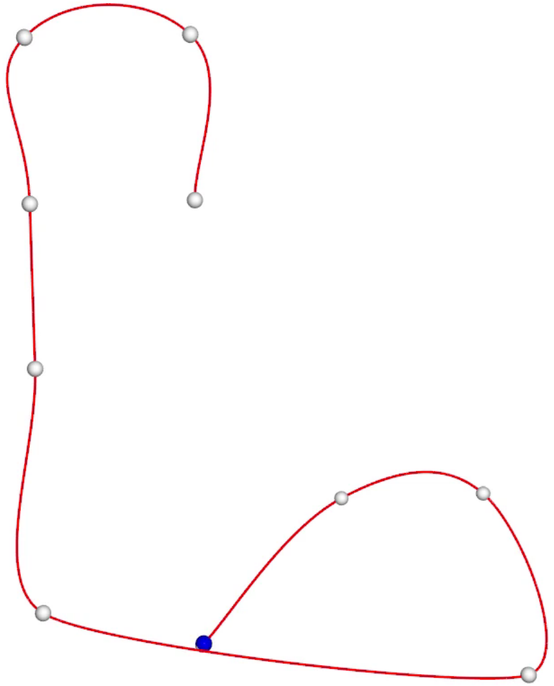
Set $d_i = \mu \dfrac{p_{i+1} - p_{i-1}}{t_{i+1} - t_{i-1}}$

- μ curve tension $\in [0, 2]$
- $\mu = 1$ is commonly used

Catmull Rom Spline



Wrap-up Algorithm



Compute $p(t)$ as a cubic spline interpolation

- Given keyframes $(p_i, t_i)_{i \in [0, N-1]}$
- Given time $t \in [t_1, t_{N-2}]$

1. Find i such that $t \in [t_i, t_{i+1}]$
2. Compute $d_i = \mu \dfrac{p_{i+1} - p_{i-1}}{t_{i+1} - t_{i-1}}$, and $d_{i+1} = \mu \dfrac{p_{i+2} - p_i}{t_{i+2} - t_i}$
3. Compute $p(t) = (2s^3 - 3s^2 + 1)\, p_i + (s^3 - 2s^2 + s)\, d_i + (-2s^3 + 3s^2)\, p_{i+1} + (s^3 - s^2)\, d_{i+1}$
   with $s = \dfrac{t - t_i}{t_{i+1} - t_i}$.

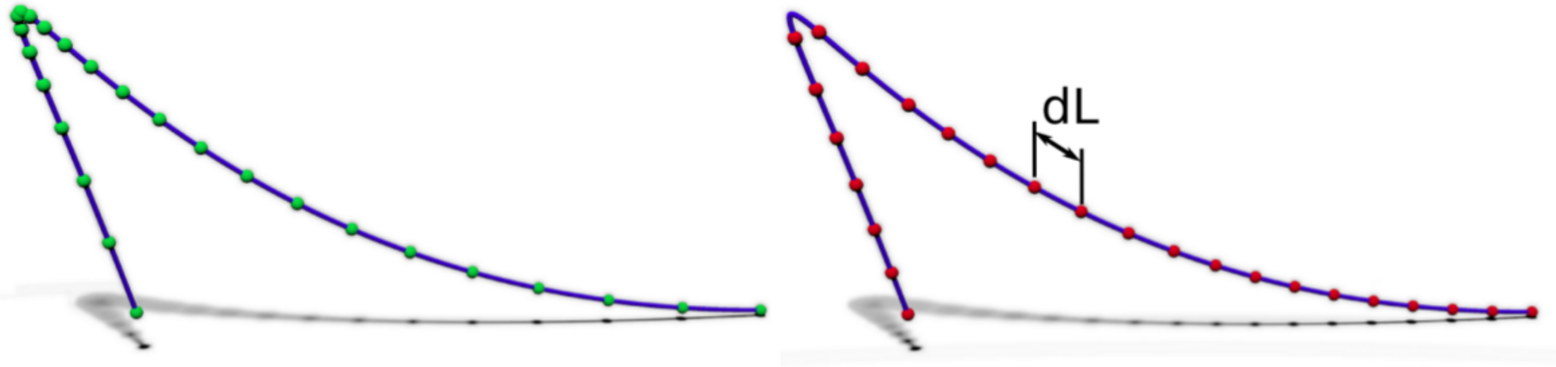
Limitation of cubic curve interpolation

- Only C^1, but not C^2 at junctions : Curvature/acceleration discontinuity

  - *Force C^2 continuity for cubic polynomial (global linear system to solve - loose local structure)*
  - *Consider higher degree polynomial*

- Non constant speed along each polynomial

  - *Reparametrization with curvilinear length*

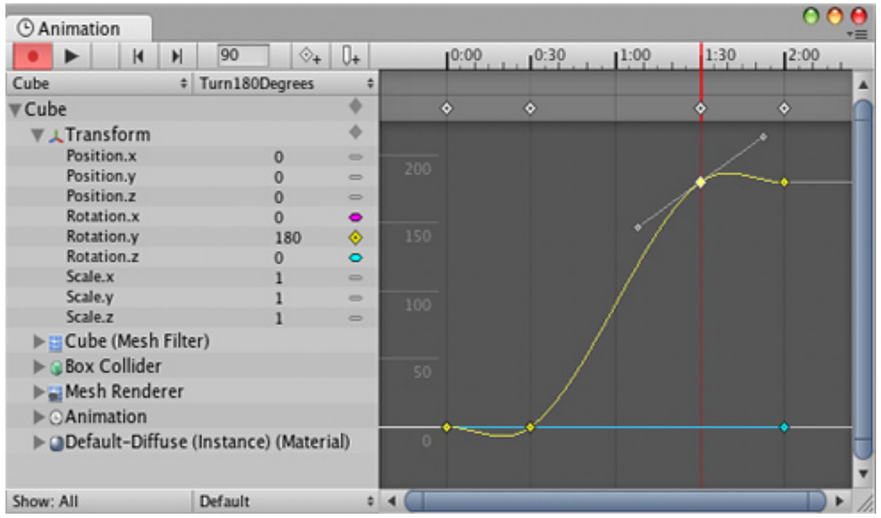


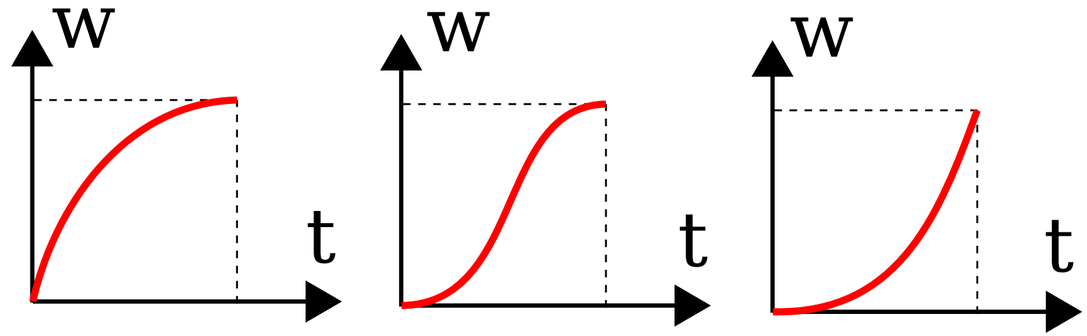
Curve editing

- Animation software (Maya, 3DSMax, Blender, etc) always come with a **curve editor**.

- Artists can manually ajust their position, time, and **derivatives** on curve editor.
  - One curve for each scalar parameter
    - position (x, y, z)
    - scaling (s_x, s_y, s_z)
    - rotation/quaternion (q_x, q_y, q_z, q_w)

- Can also use a wrapper function w to change time $p(t) = f(w(t))$





Usage of keyframes interpolation

Interpolate every vertex of multiple meshes

Multi-target blending

Interpolate between multiple key poses

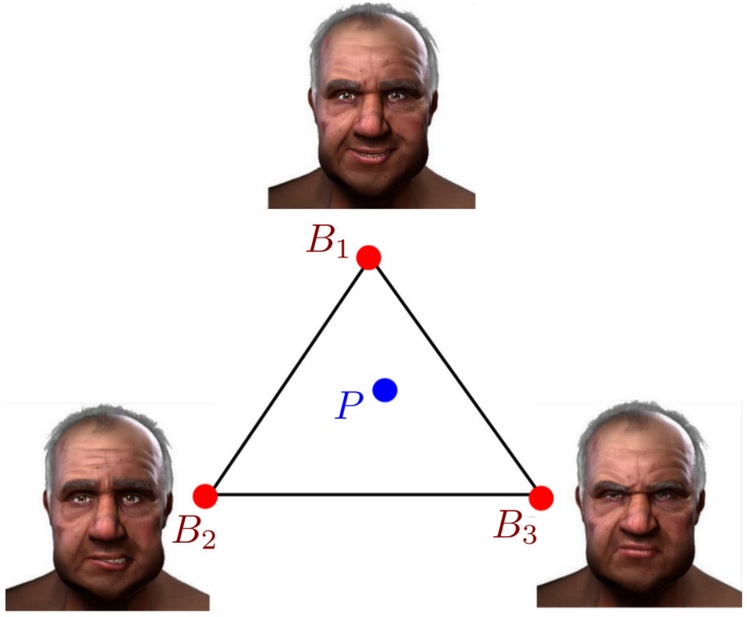
Interesting for facial animation

ex. 30% happy, 50% suprised, 20% tired

- Per-vertex formulation $p_i(t) = \sum_{k}^{N_{poses}} \omega_k(t)\, b_{ik}$

  - p_i: ith deformed vertex coordinates
  - b_{ik}: Initial ith vertex coordinates at pose k
  - w_k: Interpolation weight for pose k.

- Matrix formulation $\mathbf{p}(t) = \mathrm{B}\,\omega(t)$



Sometimes easier to control in adding "change of expression" instead of interpolating.

ex. Neutral + 20% happy smile + 30% surprised eyes

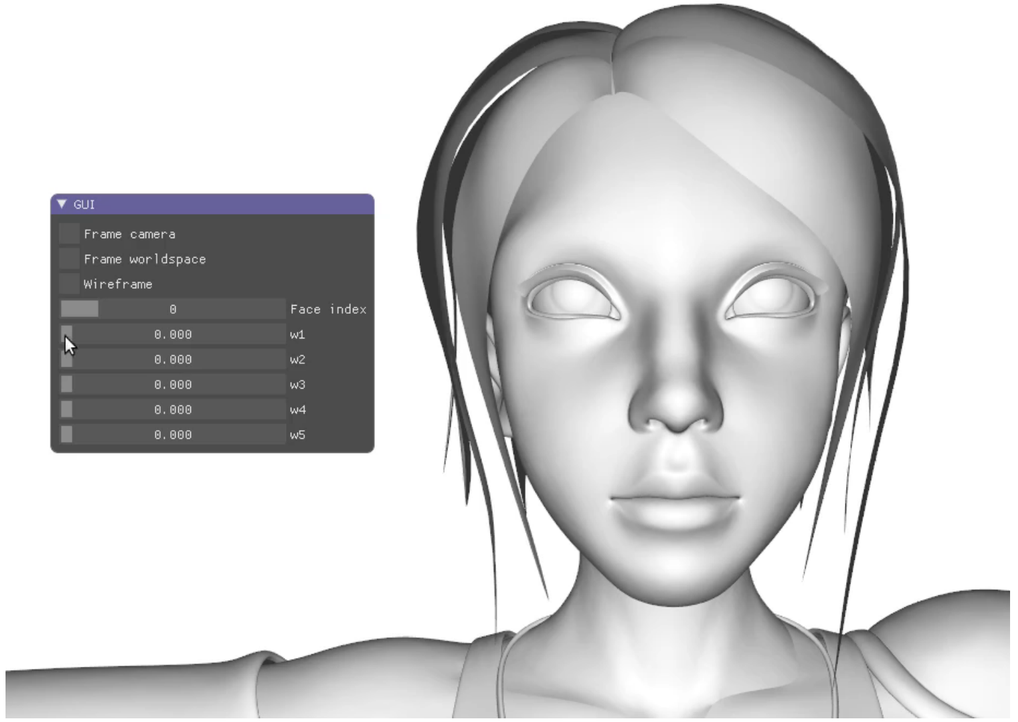
Blend shapes

Represent deformation as a difference of coordinates with respect to neutral pose

- Per-vertex formulation

$$p_i(t) = b_i^0 + \sum_k^{N_{poses}} \omega_k(t) \underbrace{(b_{ki} - b_i^0)}_{d_{ki}}$$

- Matrix formulation $\mathbf{p}(t) = \mathbf{b^0} + \mathrm{D}\,\omega(t)$

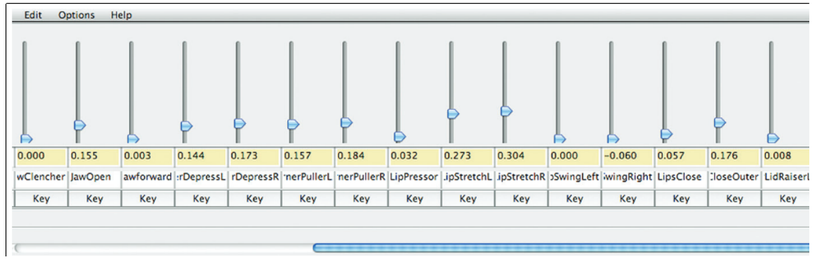


Can add/substract local expressions (even extrapolate)

as long as they are not interfering/redundant

Allows expression transfert (keep same D for different faces b^0).

same connectivity required



Note:

- D is not an orthogonal matrix (non unique combination, redundancies) $\Rightarrow$ decomposition PCA, etc.
- How to directly manipulate blend shapes

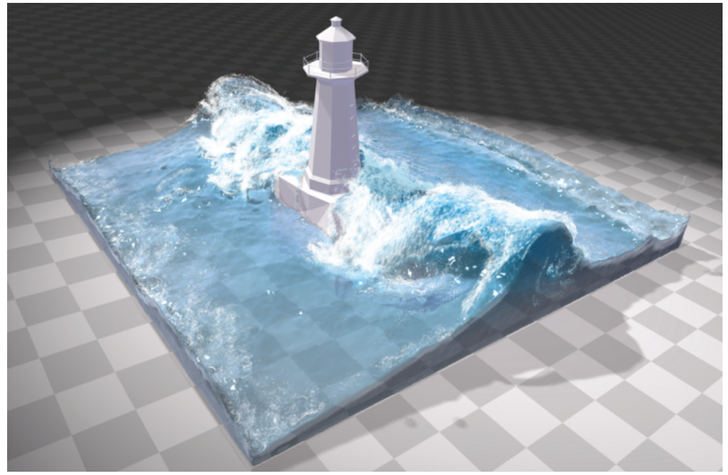
J.P. Lewis et al. Practice and Theory of Blendshape Facial Models. EG STAR 2014

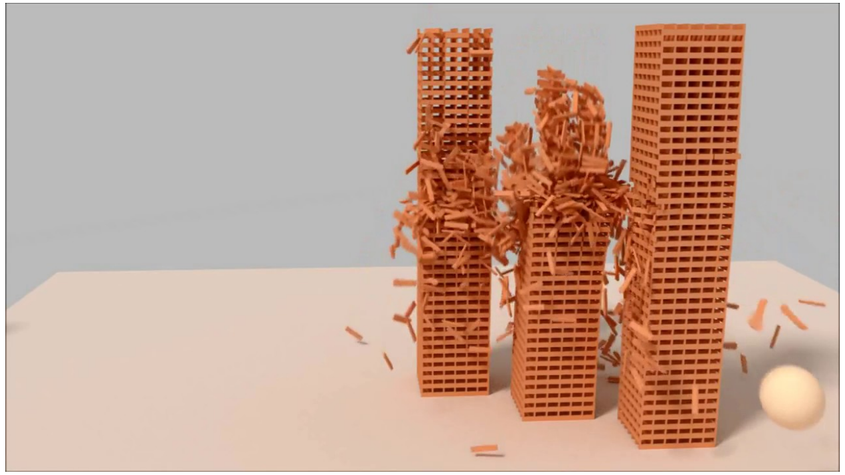
Physically-based simulation

Deformable models

When physically based simulation is needed

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





Material model

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

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

Plasticity: Oppositve of elasticity. Plastic material don'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

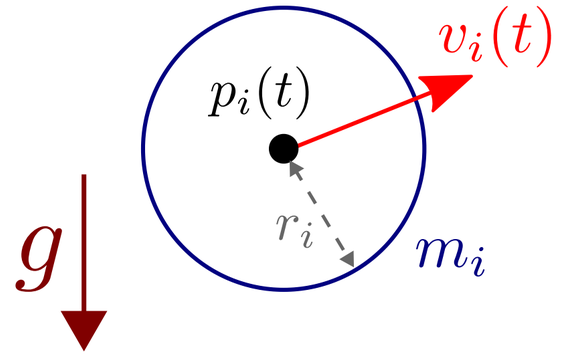
- *Elasto-plastic materials*: Allow elastic behavior for small deformation, and plastic at larger one.
- *Visco-elastic materials*: Elastic properties with delay.

Rigid spheres

System modeling

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, speed 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

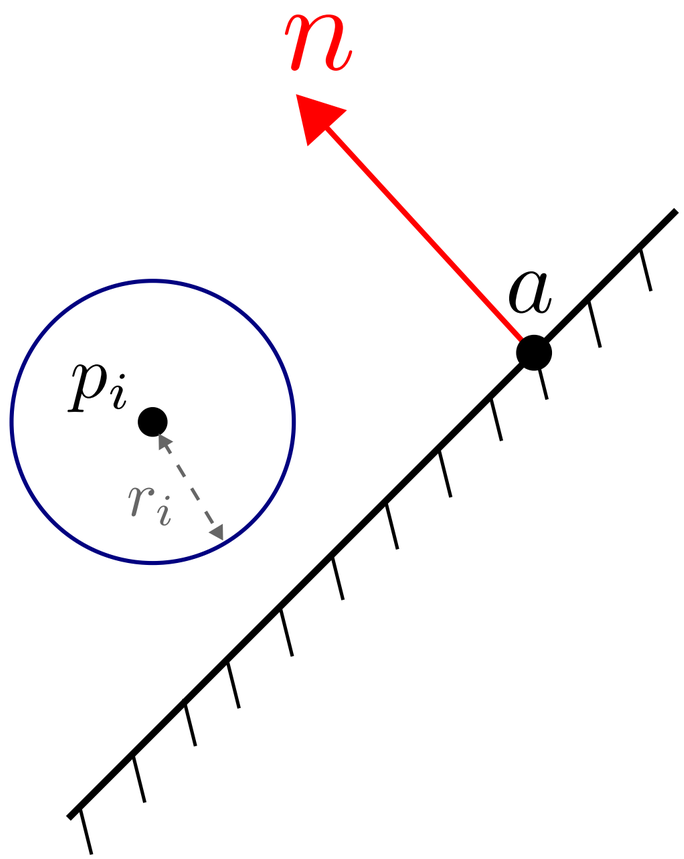
Plane $\mathcal{P}$: parameterized using a point a and its normal n.

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

- 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

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

Collision response = **Update speed**

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

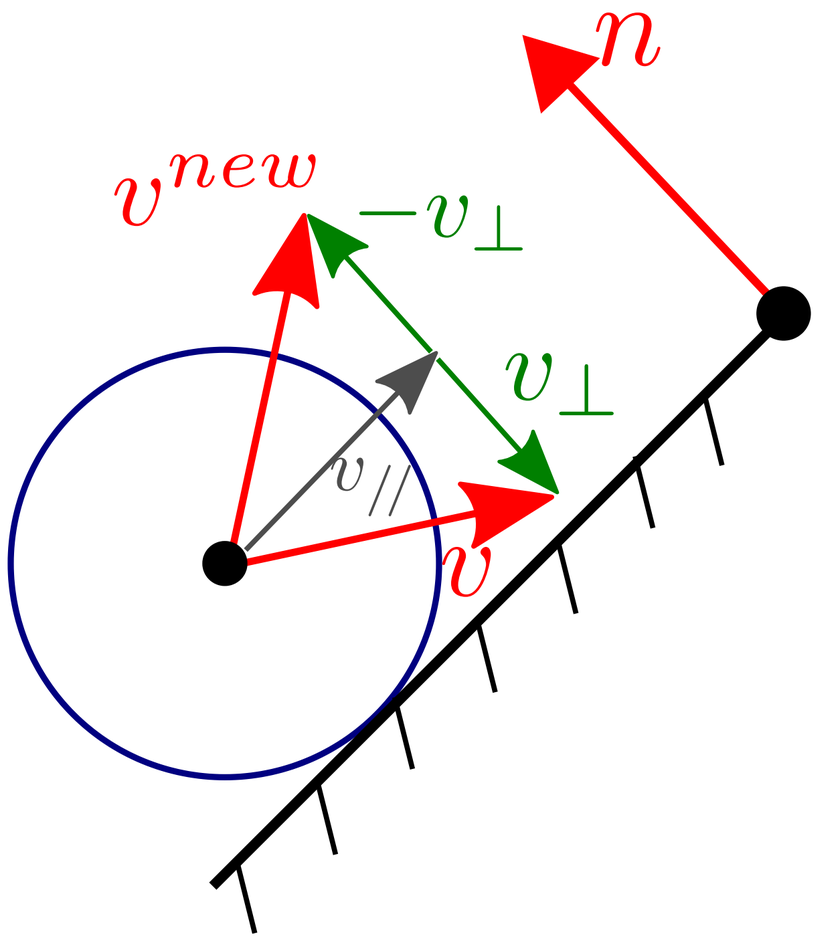
- $v_{\perp} = (v \cdot n)\, n$
- $v_{//} = v - (v \cdot n) n$

New speed

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

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

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



Result: Collision response

Applying collision response on speed only

Collision response with plane : position

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 constaint

(+) Simple to implement

(-) Physically incorrect position

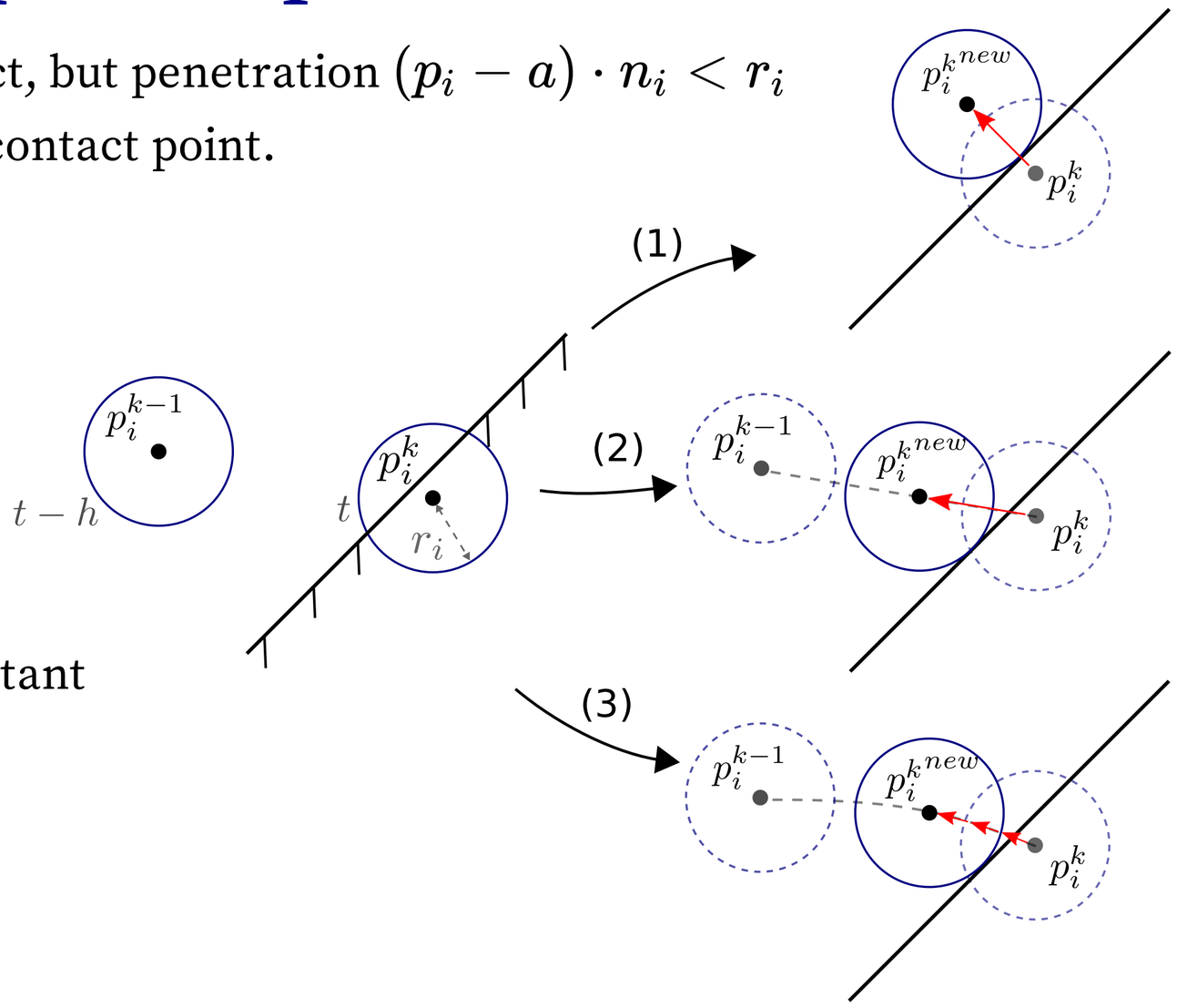
(2) Approximate the correct position

(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

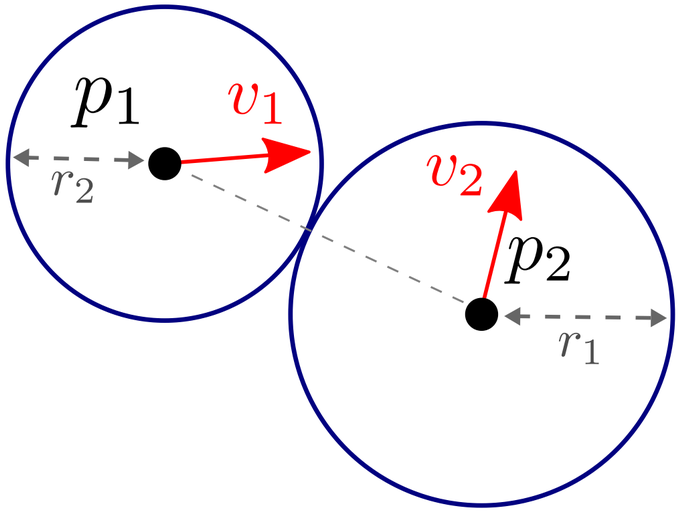
$p_i^{new} = p_i + d\,n$

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

Collision between spheres

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 will happen with speeds ?

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

Notion of impulse

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

$\rightarrow$ results in a sudden change of speed (/momentum) in a discrete case

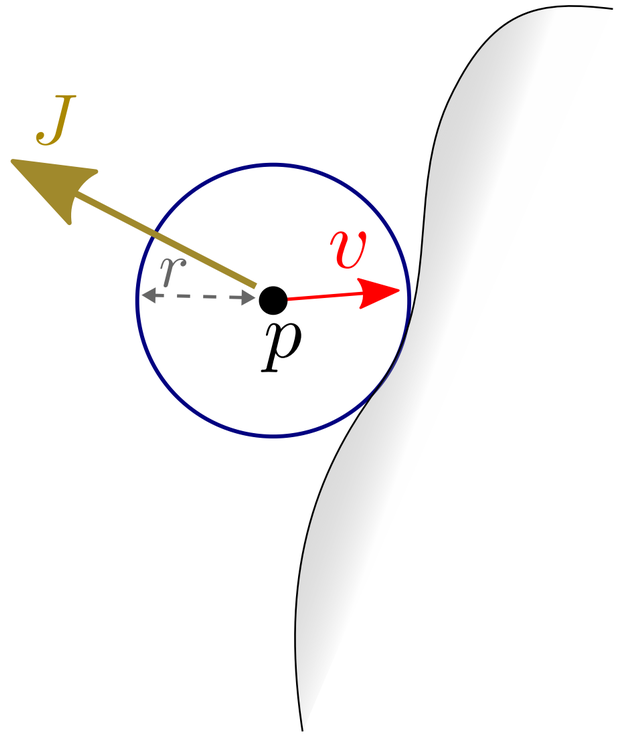
For a particle with constant mass

$$\int_{t_1}^{t_2} F(t)\,\mathrm{d}t = \int_{t_1}^{t_2} m\,a(t)\,\mathrm{d}t$$

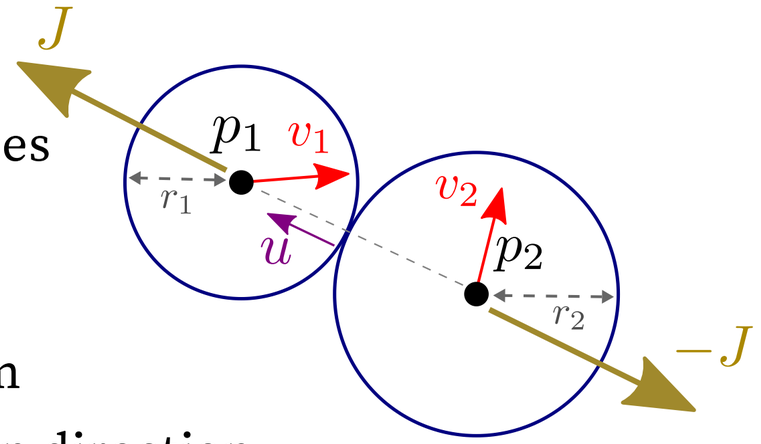
$$\Rightarrow J = m\,(v(t_2) - v(t_1))$$

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

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



Two spheres in collision



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

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

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

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

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

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

Two spheres in collision

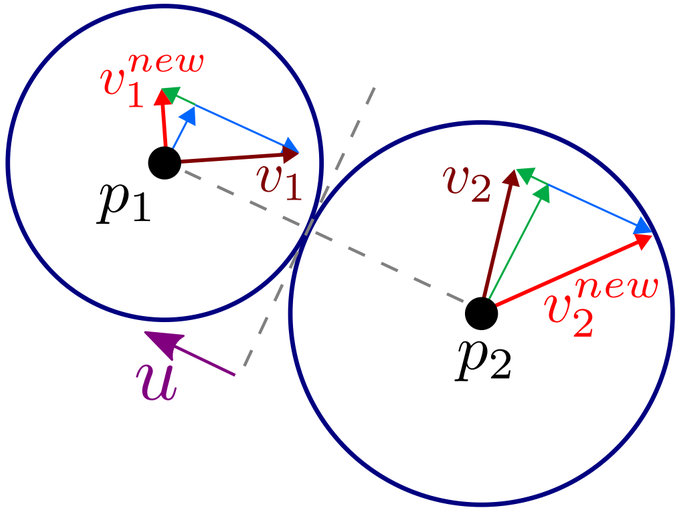
$$v_1^{new} = v_1 + j/m_1\, u = v_1 + 2\frac{m_1}{m_1+m_2}((v_2 - v_1)\cdot u)\, u$$
$$v_2^{new} = v_2 - j/m_2\, u = v_2 - 2\frac{m_1}{m_1+m_2}((v_2 - v_1)\cdot u)\, u$$

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

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

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

$$v_1^{new} = \alpha\, v_{1//} + \beta\, v_{2\perp}$$
$$v_2^{new} = \alpha\, v_{2//} + \beta\, v_{1\perp}$$



Summary

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

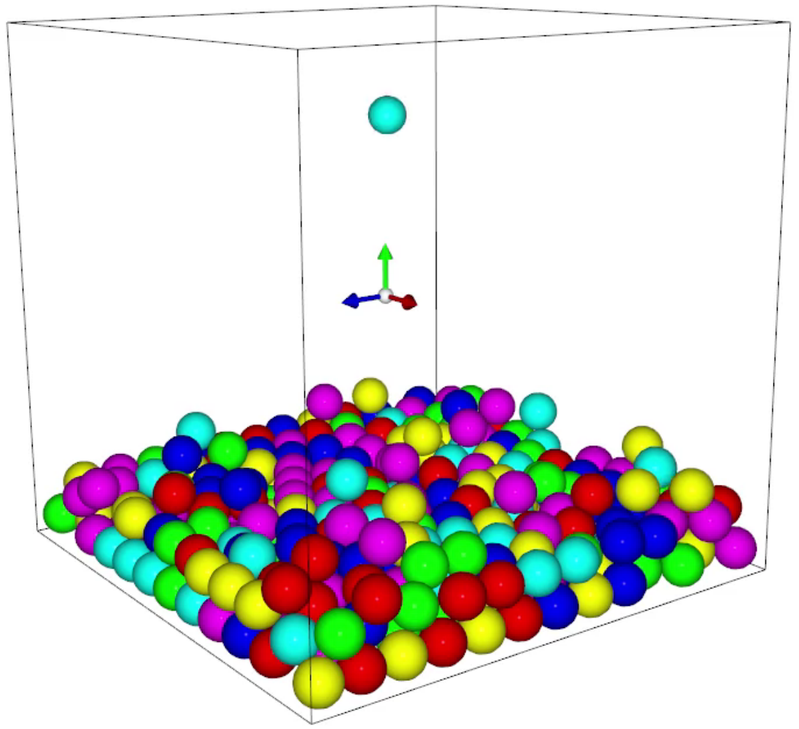
2-a. If collision (relative speed $> \epsilon$)

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

Avoids jittering



3. Correct position (project on contact surface)

$p = p + d/2\, u$

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

3-b. For small impacts, can use *position based dynamics*

$v^{new} = (p^{new} - p^{prev})/\Delta t$

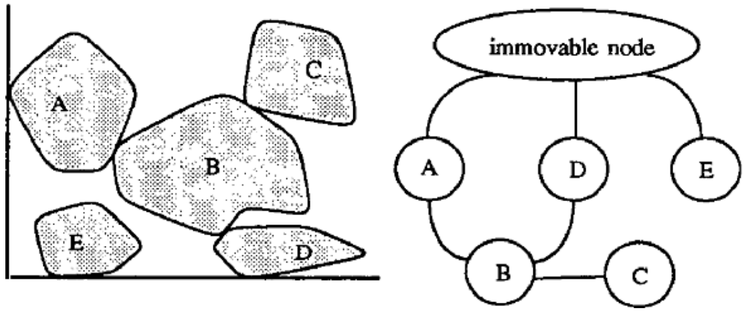
Note on collision stack

Solving pairwise collisions doesn't ensure global collision free state

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

Possible improvements

- Reducing time step
- Iterating over multiple collision resolution steps
- Global collision solving
  - (ex. Handle pile of objects explicitely)



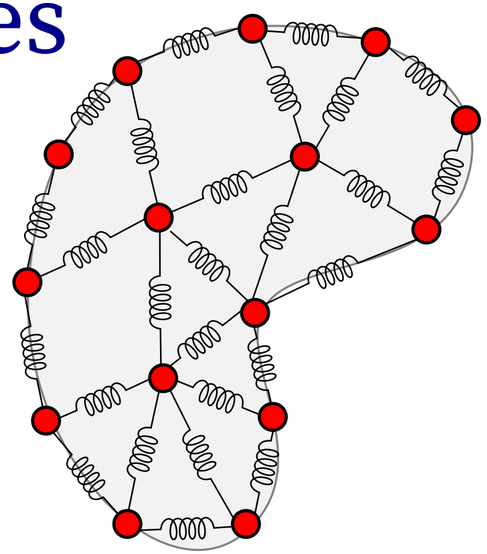
Realistic Animation of Rigid Bodies. J. Hahn. SIGGRAPH 1988.

Reflections on Simultaneous Impact. B. Smith et al. SIGGRAPH 2012

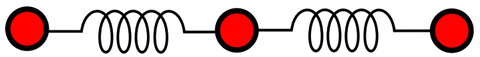
Modeling elastic shapes with particles

Spring mass systems

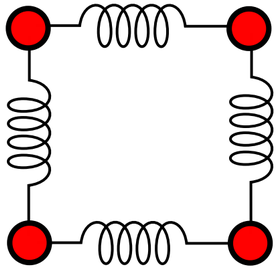
- Particles (position, speed, mass): samples on shape
- Springs : link closed-by particles in the reference shape



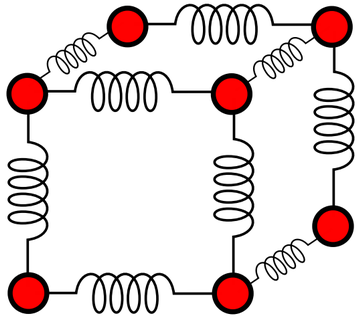
1D curve structure



2D surface structure



3D volume structure

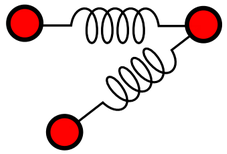


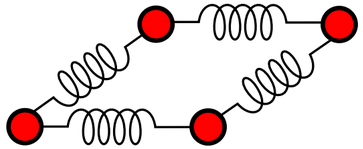
Spring structure

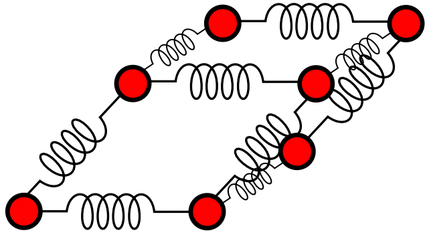
How to model spring connectivity ?

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

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

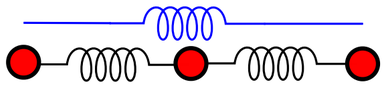


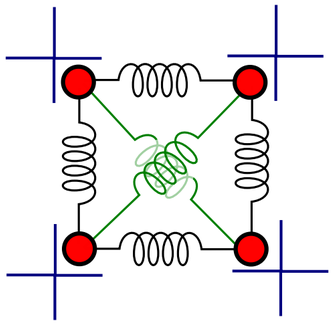


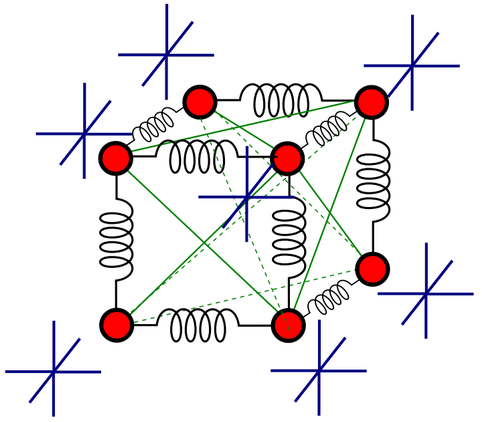


$\Rightarrow$ Add extra springs connectivity

- **Shearing springs**: Diagonal links
- **Bending springs**: 2-ring neighborhood

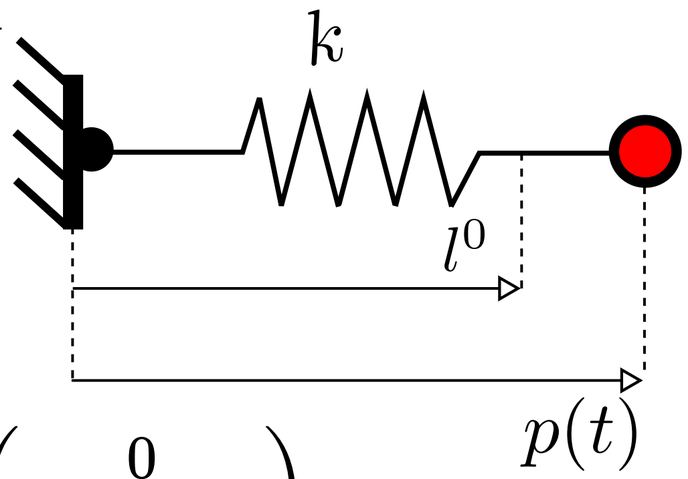






Example: 1D spring (/Harmonic oscillator)

- Force $F(t) = -k\,(p(t) - l^0)$, k spring stiffness, l^0 rest length
- Second order differential equation: $m\,p''(t) + k\,p(t) = k\,l^0$



- First order system $\underbrace{\begin{pmatrix} p \\ v \end{pmatrix}'(t)}_{u'(t)} = \underbrace{\begin{pmatrix} v(t) \\ -k/m\,(p(t) - l^0) \end{pmatrix}}_{\mathcal{F}(u,t)}$

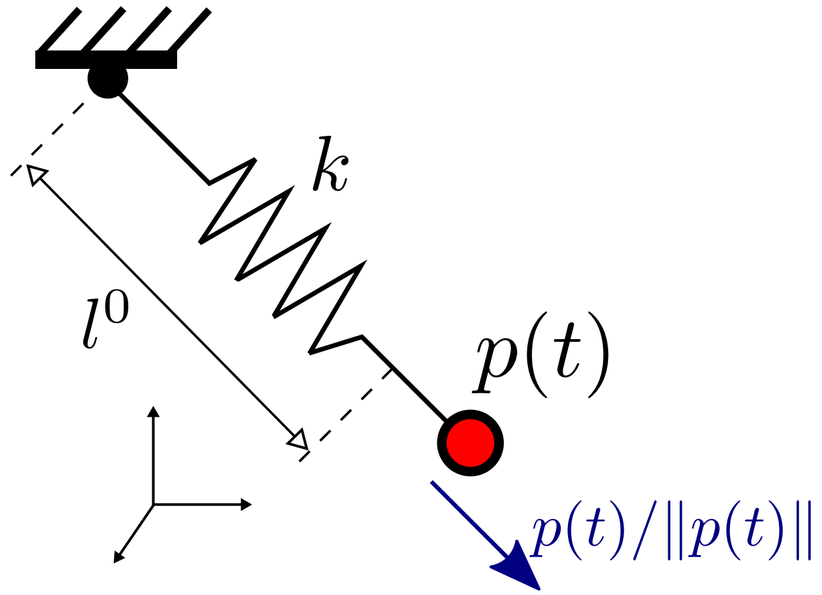
- Linear system $\underbrace{\begin{pmatrix} p \\ v \end{pmatrix}'(t)}_{u'(t)} = \underbrace{\begin{pmatrix} 0 & 1 \\ -k/m & 0 \end{pmatrix}}_{\mathrm{A}} \underbrace{\begin{pmatrix} p \\ v \end{pmatrix}(t)}_{u(t)} + \underbrace{\begin{pmatrix} 0 \\ k/m\,l^0 \end{pmatrix}}_{b}$

- Exact solution known: $p(t) = A\,\sin(\omega\,t + \varphi) + l^0$

$$\omega = \sqrt{k/m},\, A^2 = (p^0 - l^0)^2 + \left(\frac{v_0}{\omega}\right)^2,\, \tan(\varphi) = \frac{p^0 - l^0}{v^0/\omega}$$

Example: 3D mass spring

- Force $F(t) = m\,g - k\,(\|p(t)\| - l^0)\,\frac{p(t)}{\|p(t)\|}$, k spring stiffness, l^0 rest length
- Second order differential equation: $m\,p''(t) = mg - k\,(\|p(t)\| - l^0)\,\frac{p(t)}{\|p(t)\|}$
- First order system $\underbrace{\begin{pmatrix} p \\ v \end{pmatrix}'(t)}_{u'(t)} = \underbrace{\begin{pmatrix} v(t) \\ g - k/m\,(\|p(t)\| - l^0)\,\frac{p(t)}{\|p(t)\|} \end{pmatrix}}_{\mathcal{F}(u,t)}$
- Not linear
- No simple explicit solution



Numerical integration of ODE

General formulation: $u'(t) = \mathcal{F}(u,t), u(t) = (p(t), v(t))$.

Explicit Euler

$u^{k+1} = u^k + \Delta t\, \mathcal{F}(u^k, t^k)$

(+) Easy to implement

(-) Worst scheme in all cases (divergence, low accuracy)

Explicit Runge-Kutta

$u^{k+1} = u^k + \Delta t \sum_j \alpha_j k_j$

(+) Good **accuracy**

(+) Efficient to apply

(+/-) Stability OK for non-stiff problem, diverge on stiff problem

(-) Artificial damping for constant energy system

Implicit methods

$u^{k+1} = u^k + \Delta t\, \mathcal{F}(u^{k+1}, t^{k+1})$

(+) Good to deal with **stiff problem** - very stable

(-) Add numerical damping (converge even if solution oscillates)

(-) Hard/computationally costly to apply on non linear problem

Symplectic integrator

$v^{k+1} = v^k + \Delta t\, F^k/m$

$p^{k+1} = p^k + \Delta t\, v^{k+1}$

(+) Handle well constant energy system, preserves energy (Hamiltonian systems)

(+) Simple and efficient to implement

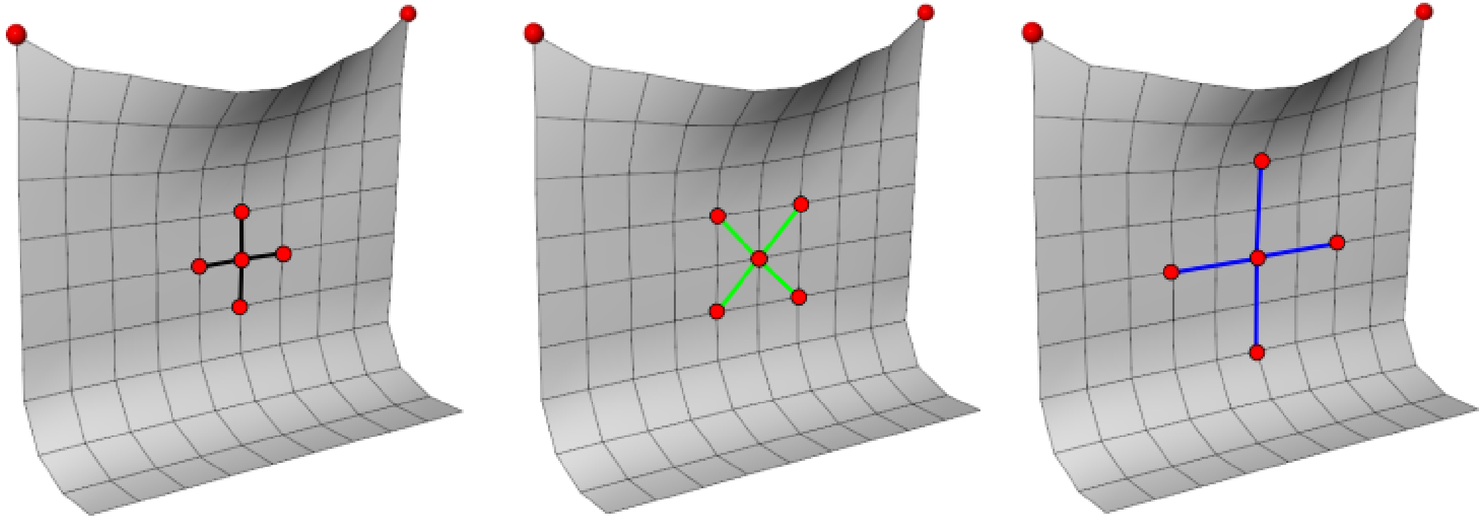
(-) Less accurate than RK

(-) Diverge on stiff problem

Cloth Simulation

Mass-spring cloth simulation

- 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.



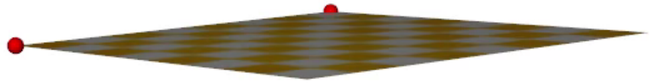
Forces

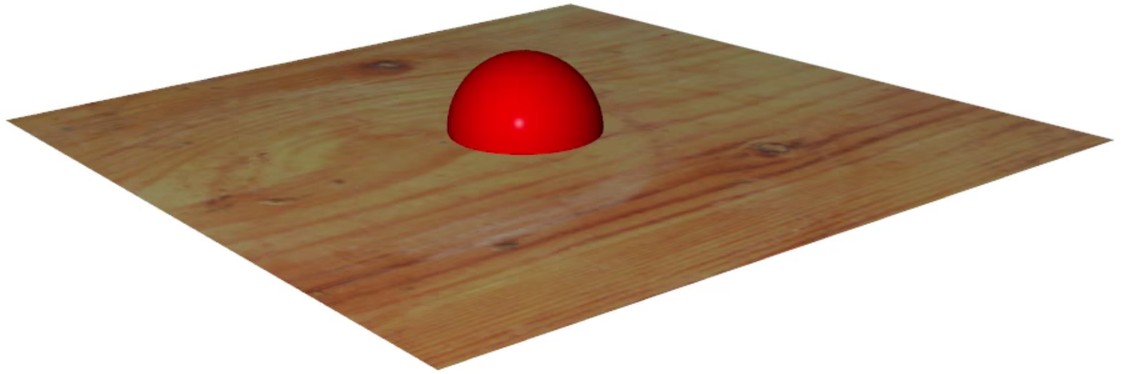
- On each particle: gravity + drag + spring forces

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

  - $\mathcal{V}_i$: neighborhood of particle i
  - L^0_{ij}: rest length of spring ij

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



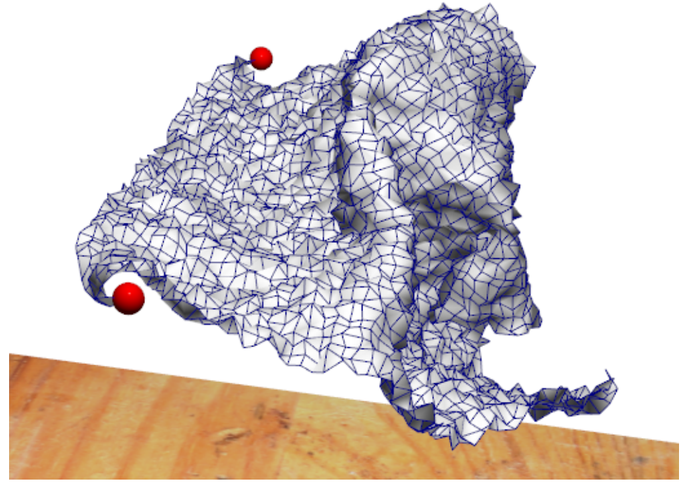


Note on Mass-Spring numerical solution

- Non-linear ODE

- Large K_{ij} : good length preservation, but stiff ODE
  $\Rightarrow$ 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

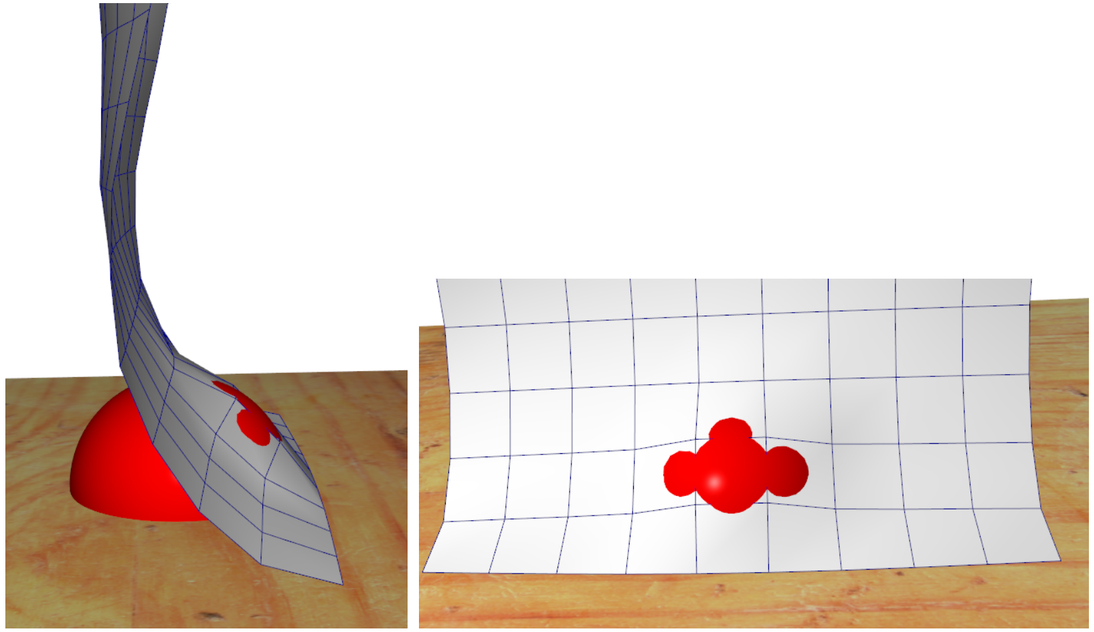
Collisions

- Simple approach : Handled as collision between particles and shapes

  (+) Simple and efficient

  (-) Collision may still appears within a triangle

  $\Rightarrow$ Exaustive approach: point-face + edge-edge



Limitation of mass spring model and continuous model

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

  $\Rightarrow$ No :(

  Depends on the connectivity $\rightarrow$ 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)

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

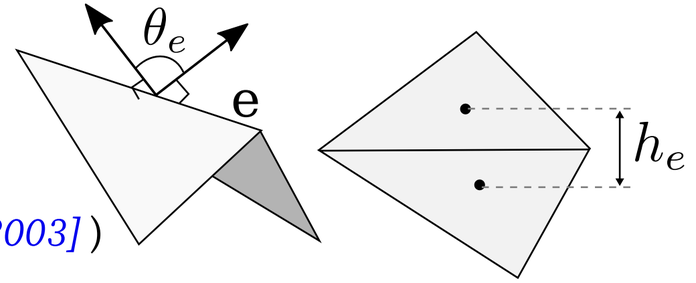
$$E = \frac{1}{2} K\, L\, \kappa^2, \kappa\text{: curvature}$$

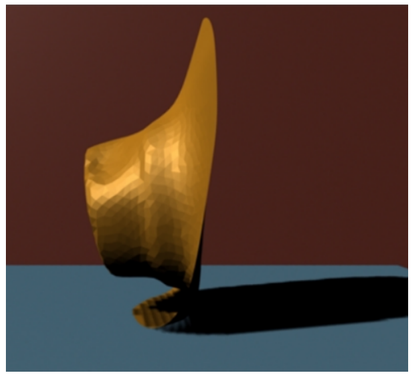
[Cho et al, Stable but Responsive Cloth, ACM SIGGRAPH 2002]

Triangle as continuous elements

- Defining Bending Energy between triangles

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

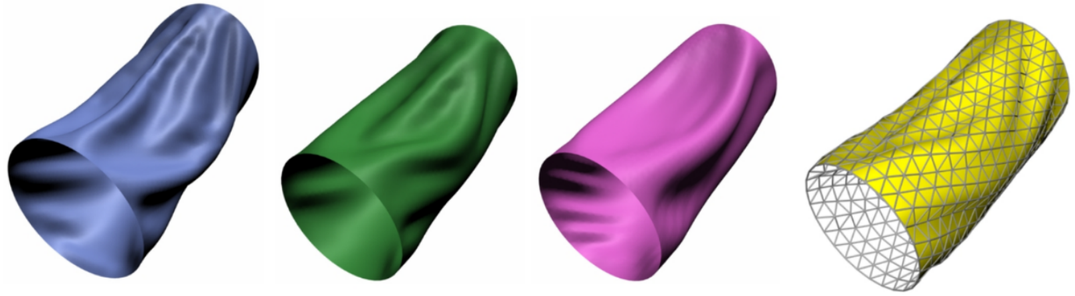




[E. Grinspun et al., Discrete Shells, SCA 2003]

(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].



Animating Virtual Characters

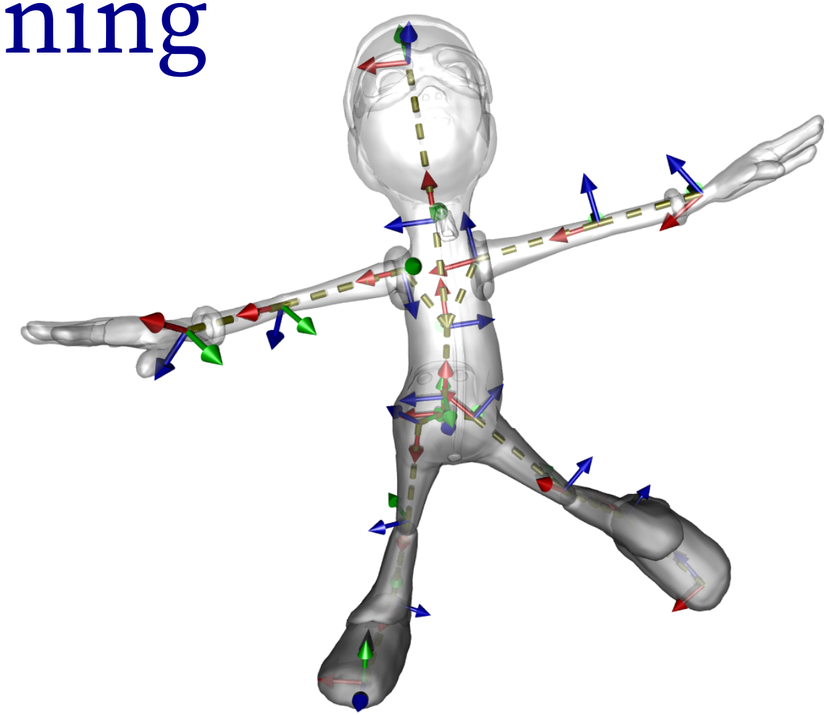
Skin deformation

Skeleton based deformation - Skinning

Objective: Deform articulated character

Idea: Use skeleton to control limbs

Articulations as rotations



Animation Skeleton

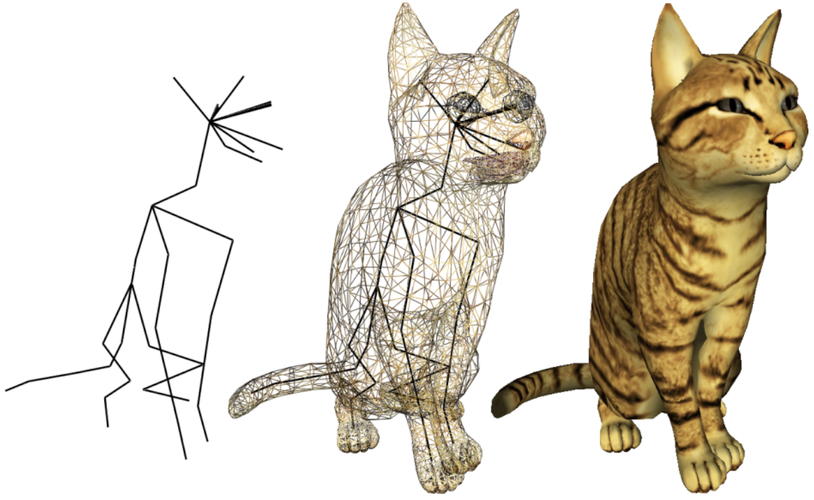
Set of frames T_i: position, orientation

Describe non-rigid parts of the character (dof)

Terminology

Joint = A frame T_i

Bone = Segment between two joints

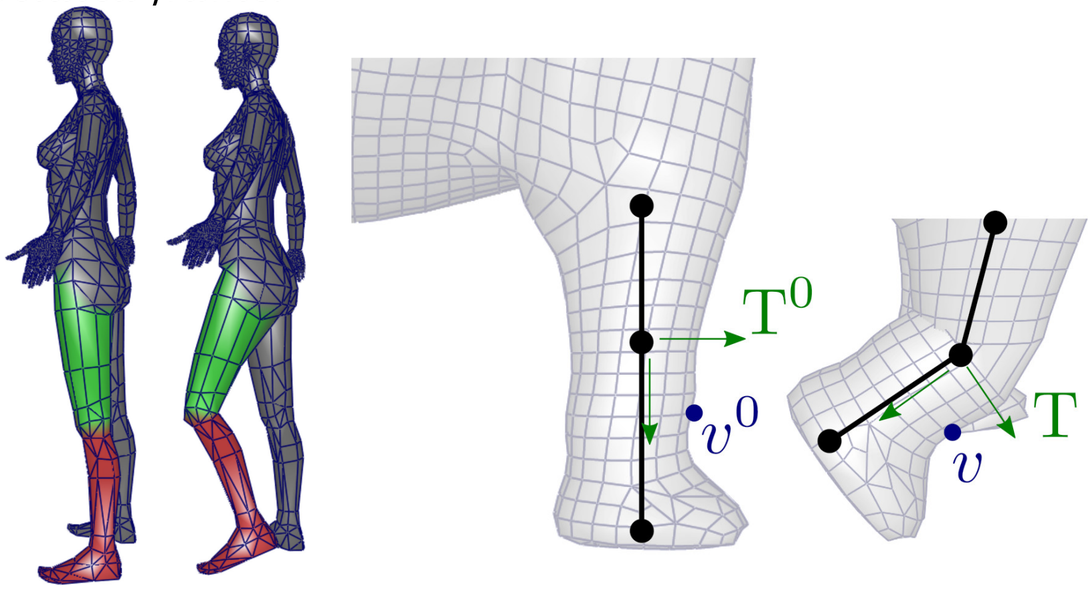


Note: Animation skeleton $\neq$ Anatomical skeleton

Rigid skinning

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

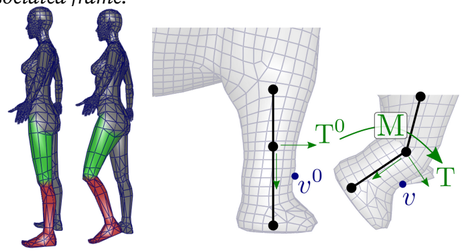
- New frame T
- New vertex position v

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

Rigid skinning

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

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

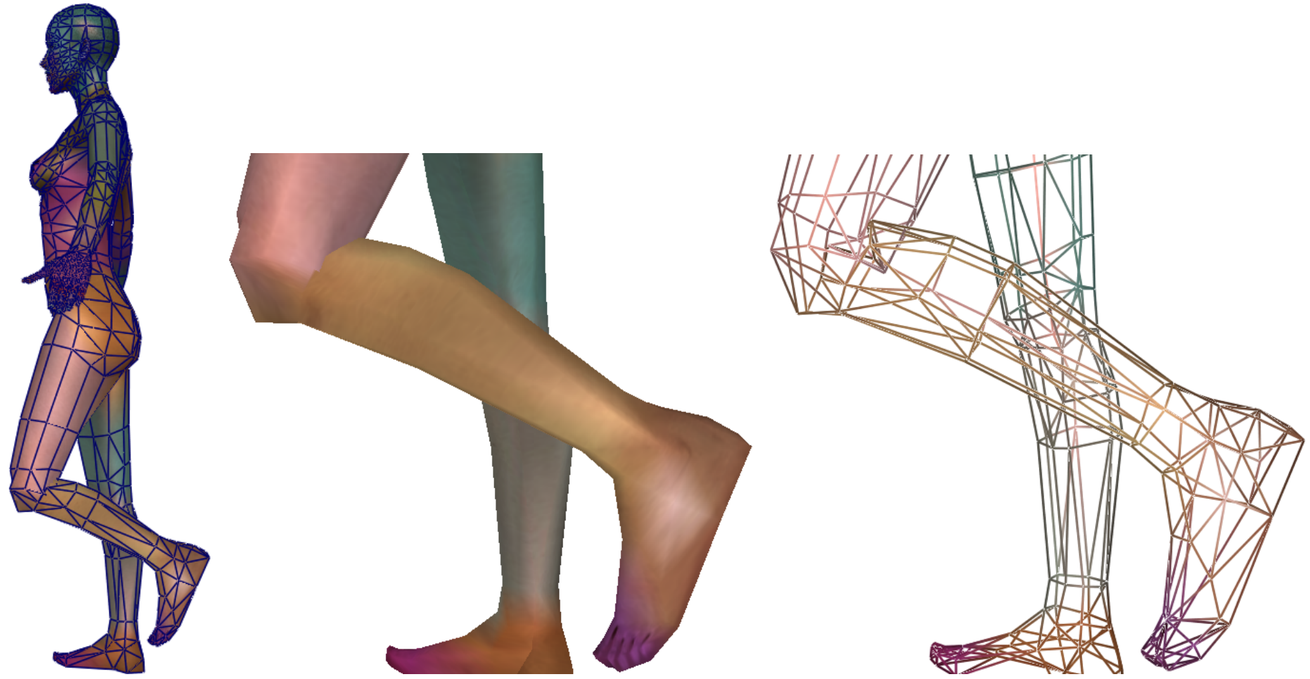
$$\Rightarrow v = \mathrm{T}\,(\mathrm{T}^0)^{-1} v_0 = \mathrm{M}\, v_0$$

Rigid Skinning

(+) Skeleton is easy to build

(+) Skeleton interaction is intuitive to model rigid articulation

(-) Discontinuities/Inter-penetrations



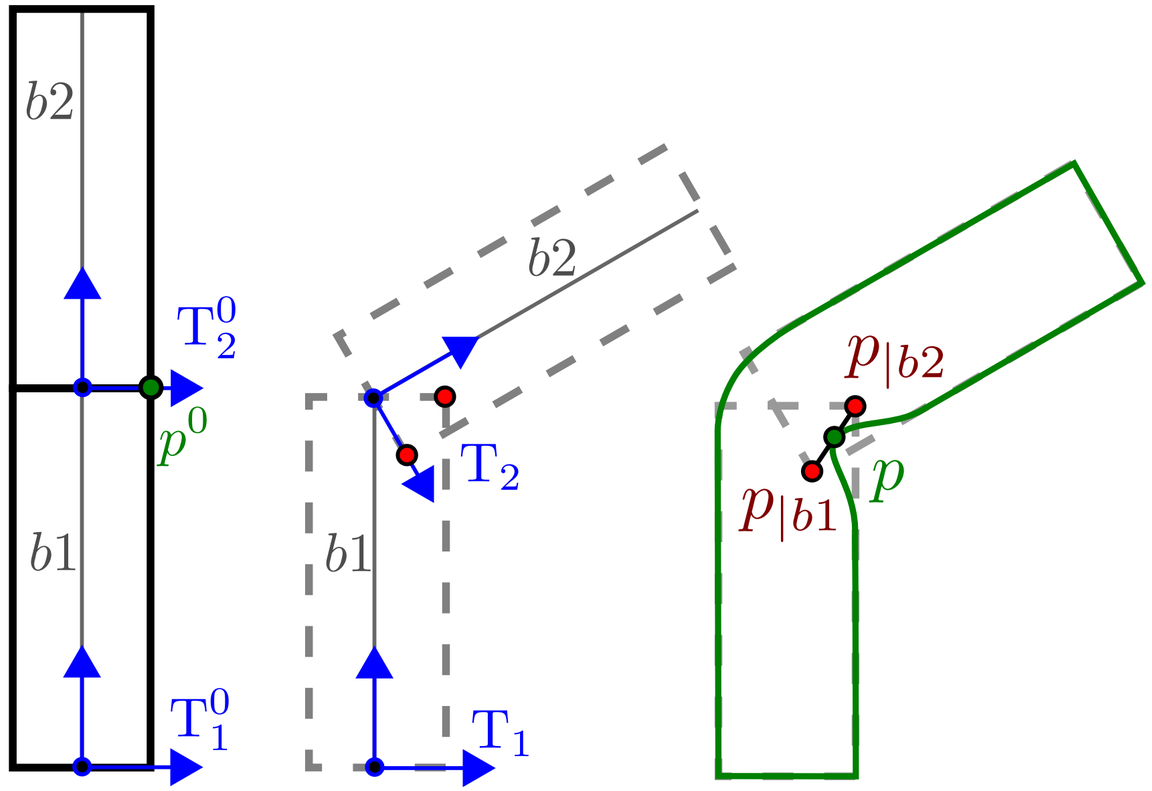
Idea of smooth skinning: Blend discontinuous transformation around articulation

Smooth skinning

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

Example at middle vertex position of a bending cylinder

$$p = 0.5\, p_{|b1} + 0.5\, p_{|b2}$$
$$p = 0.5 \underbrace{\mathrm{T}_1 \left(\mathrm{T}_1^0\right)^{-1}}_{\mathrm{M}_1} p^0 + 0.5 \underbrace{\mathrm{T}_2 \left(\mathrm{T}_2^0\right)^{-1}}_{\mathrm{M}_2} p^0$$
$$p = \left(0.5\, \mathrm{M}_1 + 0.5\, \mathrm{M}_2\right) p^0$$



Smooth skinning

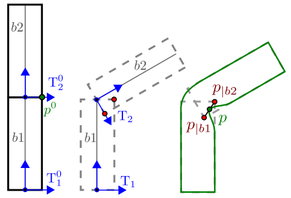
Linear Blend Skinning (LBS): Linear interpolation of positions between associated frames

Example at middle vertex position of a bending cylinder

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

$$p = 0.5 \underbrace{\mathrm{T}_1 \left(\mathrm{T}_1^0\right)^{-1}}_{\mathrm{M}_1} p^0 + 0.5 \underbrace{\mathrm{T}_2 \left(\mathrm{T}_2^0\right)^{-1}}_{\mathrm{M}_2} p^0$$

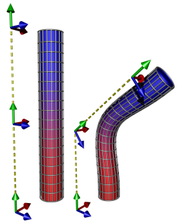
$$p = \left(0.5\, \mathrm{M}_1 + 0.5\, \mathrm{M}_2\right) p^0$$



Can be generalized to arbitrary interpolation between two bones

$$p = \alpha\, p_{|b1} + (1-\alpha)\, p_{|b2} = \left(\alpha \mathrm{M}_1 + (1-\alpha)\, \mathrm{M}_2\right)\, p^0$$

α: skinning weights



Can be generalized to any number of bones

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

Smooth skinning - Summary

$$p = \left(\sum_{k=0}^{N-1} \alpha_k \, \mathrm{M}_k \right) p^0 = \left(\sum_{k=0}^{N-1} \alpha_k \, \mathrm{T}_k \, (\mathrm{T}_k^0)^{-1} \right) p$$

$\forall k \, , \; \omega_k \in [0,1]$, 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

$\Rightarrow$ Heavily used in Animation cinema & Video Game

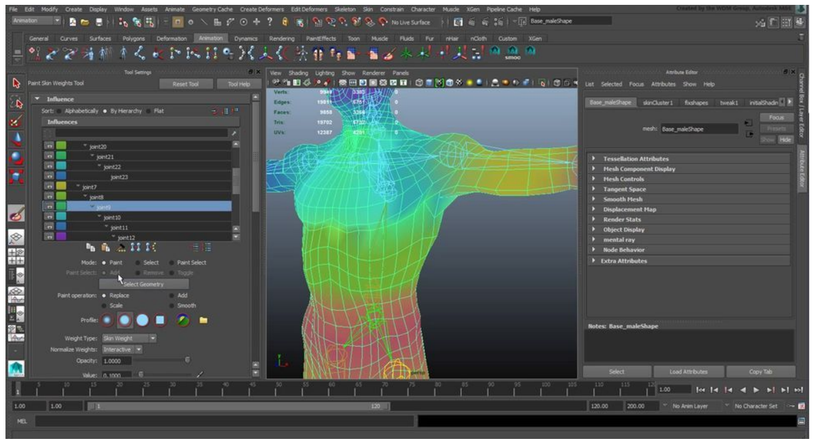
Joint-Dependent Local Deformations for Hand Animation and Object Grasping. Nadia Magnenat-Thalmann, Rochard Laperière, Daniel Thalmann. Graphics Interface, 1988

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

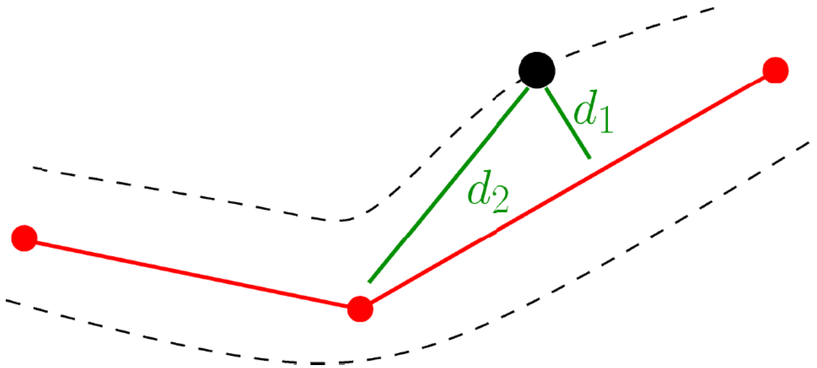
Skinning weights

How to generate skinning weights ?

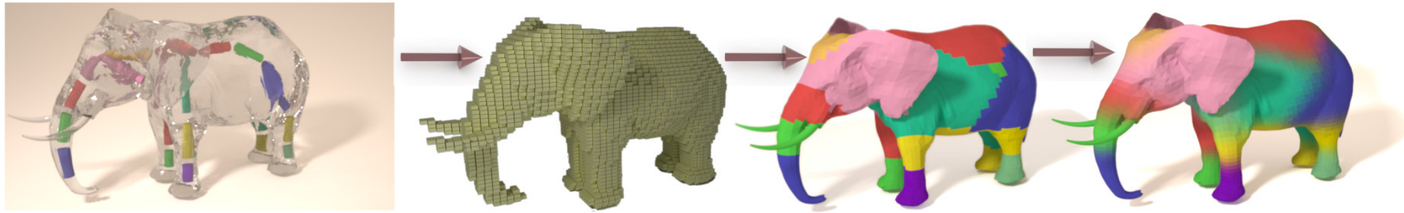
- Paint them manually
- Automatic computation



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
```


collada.dae


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


.json

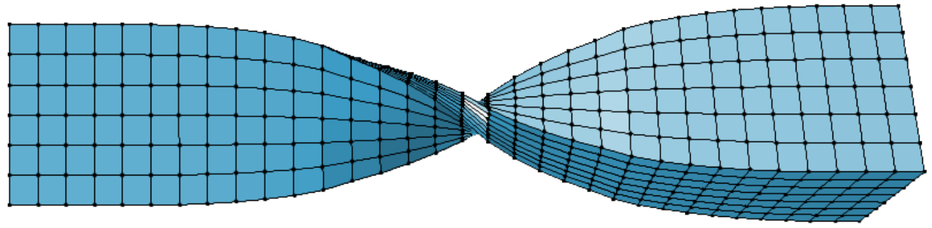
Linear Blend Skinning - Limitations

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

Candy wrapper

Collapsing elbow

Linear blending between rigid transformation matrices



Character Animation

Skeletal Animation

Skeleton structure

Characteristics

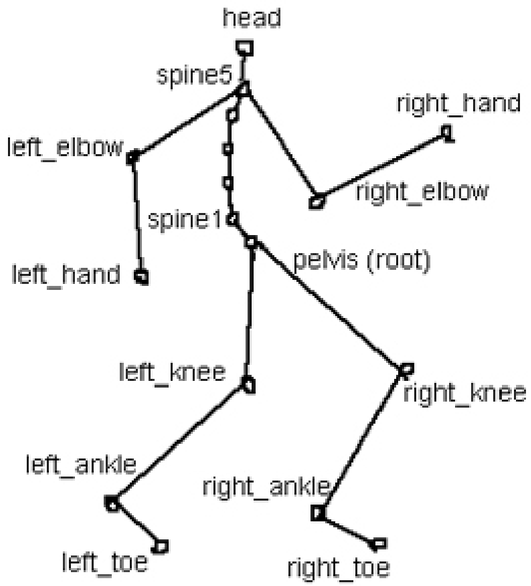
- Hierarchical representation

  All children follow the transformation of the parent

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

- Convenient to express local deformation with respect to the parent

  ex. Rotate knee from 20°



Converting local to global frames/joint coordinates

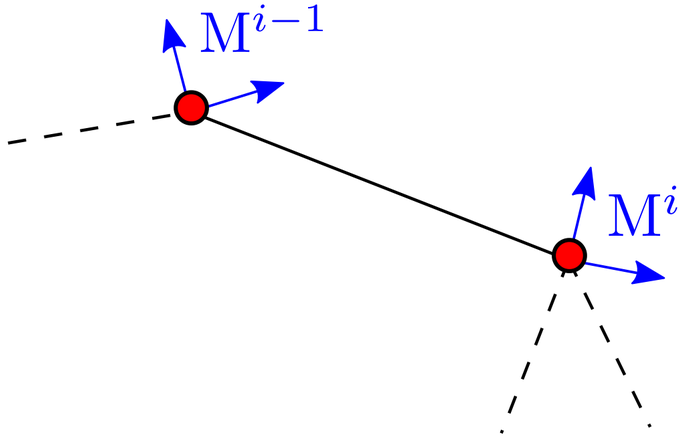
- With 4x4 matrices M

$$M^i_{global} = M^{i-1}_{global}\ M^i_{local}$$

- With translation t, rotation R

$$R^i_{global} = R^{i-1}_{global}\ R^i_{local}$$

$$t^i_{global} = t^{i-1}_{global} + R^{i-1}_{global}\ t^i_{local}$$



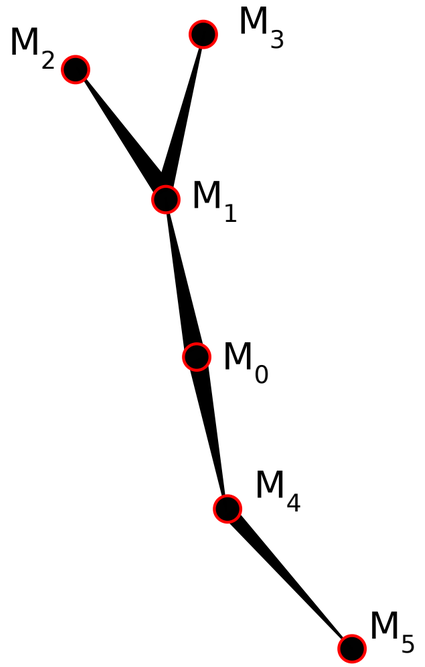
Encoding hierarchical skeleton

- 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;
}
```

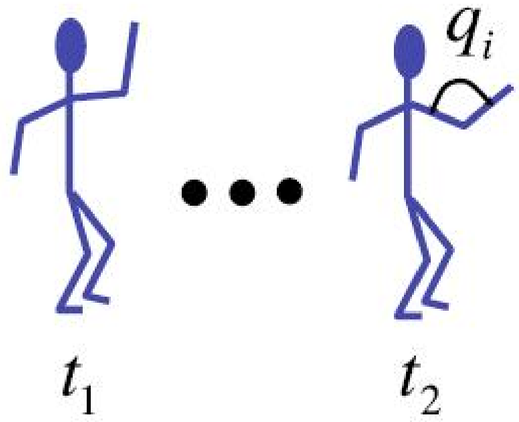


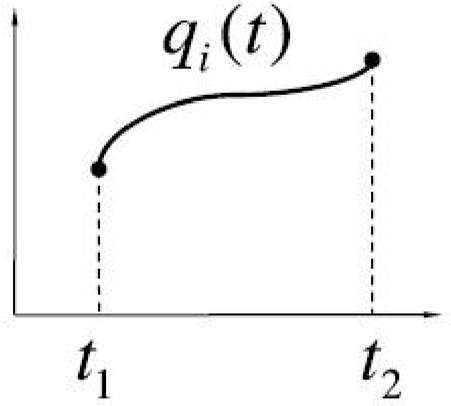
Forward kinematics

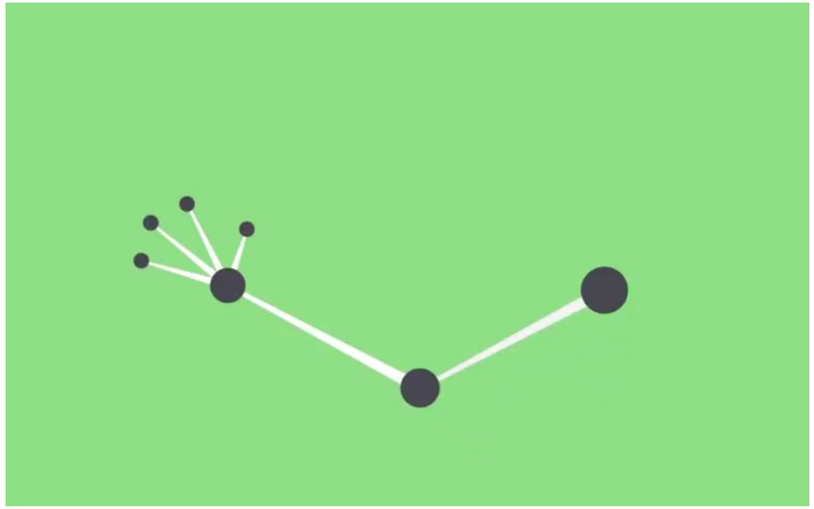
FK - **F**orward **K**inematics

- Each join angle is set manually
- Adapted to set orientation of specific parts
- Interpolate rotations during animation

(+) Generates curved trajectory naturally

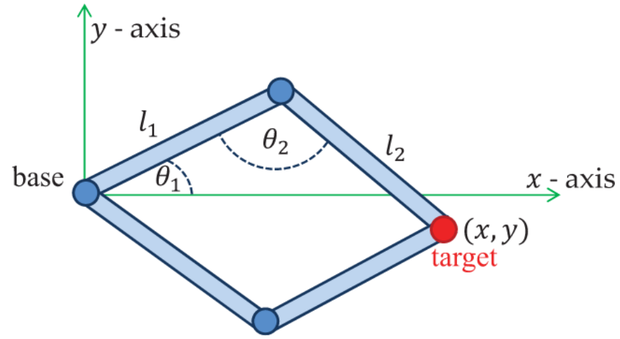






Inverse Kinematics

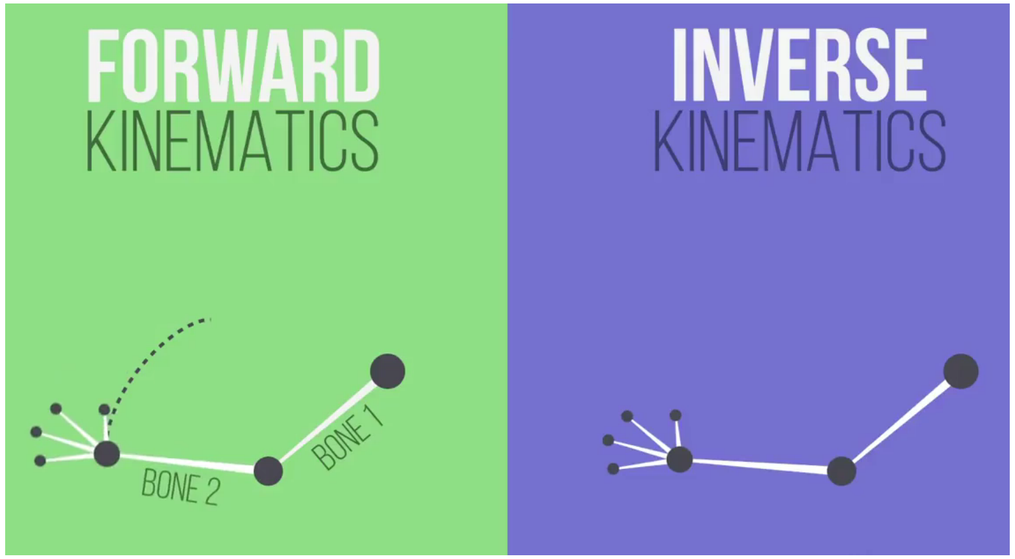
IK: Inverse **K**inematics



- Describe position (/and orientation) of *end-effectors* (contact, walking, etc)
- Compute joint angles reaching this position

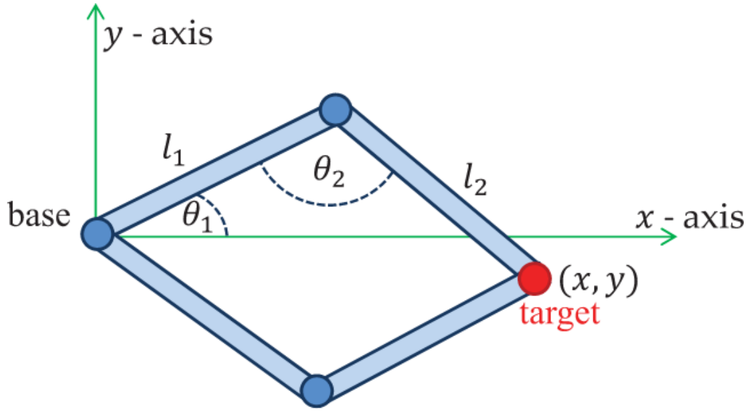
$$p_k = p_0 + \sum_{i=0}^{k-1} l_i \; R_i \, u_i$$

R_i *can be expressed with various rotation parameters (Matrices, Euler angles, axis/angle, quaternions, etc)*



MiloScerny Animation

IK Example with two bones



Inverse Kinematics Techniques in Computer Graphics: A Survey. A. Aristidou et al. STAR EG. 2017.

In general the general case $p_k = f(\theta_i)$

- Look for $\theta_i = f^{-1}(p_k)$
- f is a non linear function
- There may exists multiple solutions (or none)
- Solutions may exhibits discontinuities
- Closed form solution are not available

Two solutions defined by

$$\cos(\theta_1) = \frac{l_1^2 + x^2 + y^2 - l_2^2}{2l_1\sqrt{x^2 + y^2}}$$

$$\cos(\theta_2) = \frac{l_1^2 + l_2^2 - (x^2 + y^2)}{2l_1 l_2}$$

Some attempts for explicit solutions in specific cases

Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs. D. Tolani et al. Graphical Models, 2000.

Analytical inverse kinematics with bodyposture control. M. Kallmann. Comp. Anim. & Virt. Worlds, 2008

IK: Numerical methods

Numerical inversion of $p = f(\theta), \theta = (\theta_0, \cdots, \theta_{N-1})$.

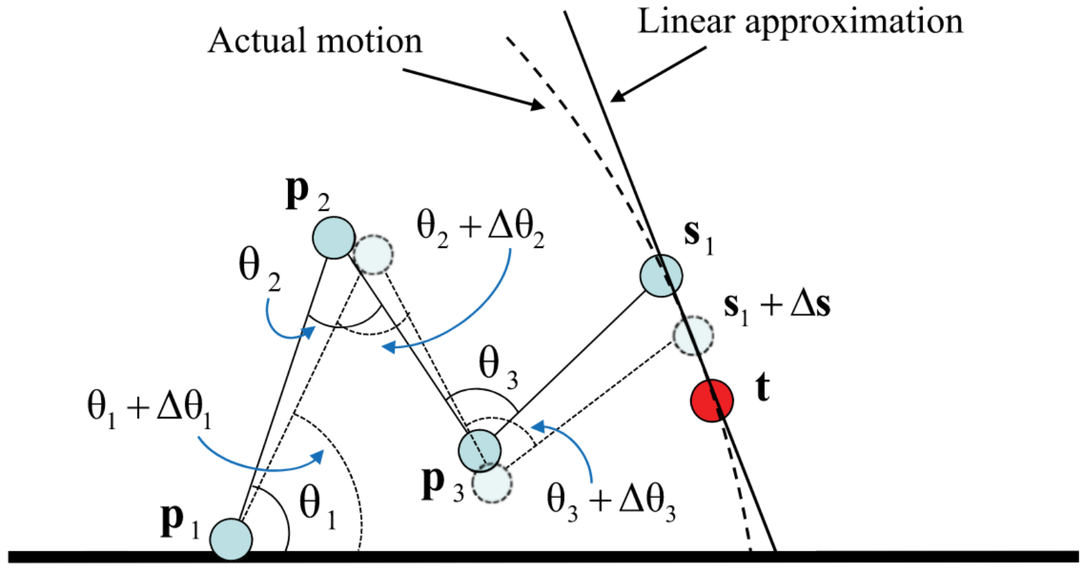
Consider small step size $p \to p + \Delta p$

$$\Delta p \simeq \underbrace{\left(\frac{\partial f}{\partial \theta}\right)}_{\mathrm{J}} \Delta\theta$$

J - Jacobian matrix.

$\to$ Not square $(3 \times N)$, not invertible.

Unknown > # constraints



IK: Numerical methods

Several possible approaches to solve $\mathrm{J}\,\Delta\theta = \Delta p$

- Pseudo Inverse

  $\Delta\theta = J^{+}\,\Delta p$, with $\quad \mathrm{J}\mathrm{J}^{+} = \mathrm{I}$

  $\mathrm{J}^{+} = \mathrm{J}^{T}\left(\mathrm{J}\,\mathrm{J}^{T}\right)^{-1}$

- Can also be computed using SVD: $\mathrm{J}^{+} = \mathrm{V}\Sigma^{+}\,\mathrm{U}^{T}$

  $\Sigma_{ii} = \sigma_i, \Sigma^{+}_{ii} = 1/\sigma_i$ if $\sigma_i \neq 0, 0$ otherwise.

- Adding damping to compensate for singularities

  $\Delta\theta = \mathrm{J}^{T}\left(\mathrm{J}\,\mathrm{J}^{T} + \lambda^{2}\,\mathrm{I}\right)^{-1}\,\Delta p$

- Using Newton's methods

Inverse Kinematics Techniques in Computer Graphics: A Survey. A. Aristidou. STAR EG 2017

IK: Heuristic approach

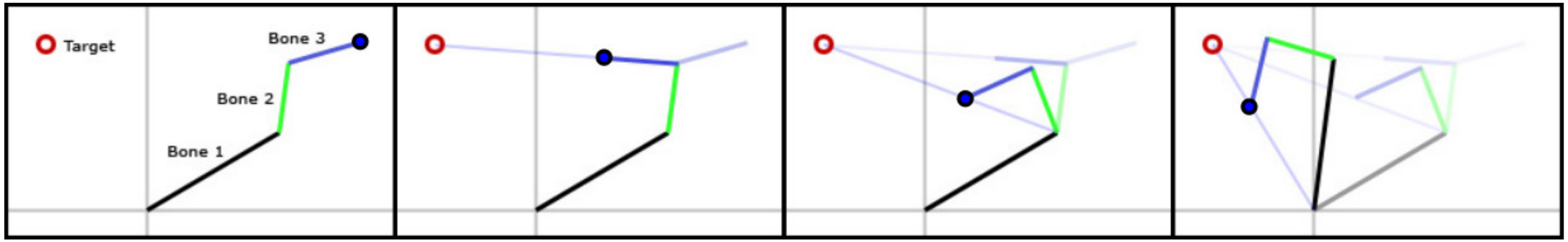
Cyclic Coordinates Descent (CCD)

- Iteratively rotates joint $j^N \rightarrow j^{i-1} \rightarrow \cdots \rightarrow j^1$ for the extremity (end effector) to be as close as possible from the target.

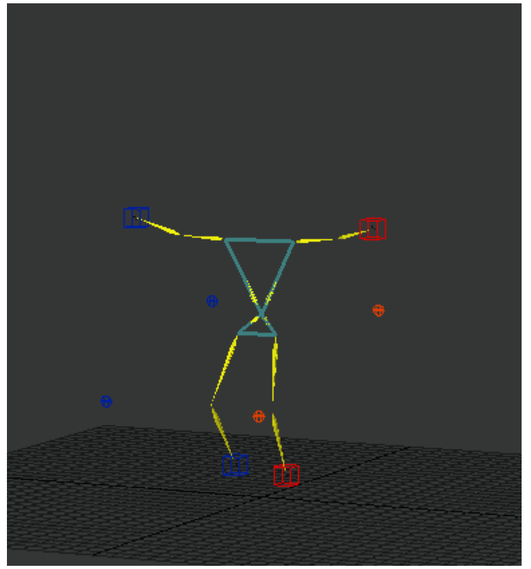
  = *End-effector aligned with the segment (joint,target)*

- Restart until convergence

Making Kine More Flexible. Jeff Lander. Game Dev, 1998.



IK: Heuristic approach

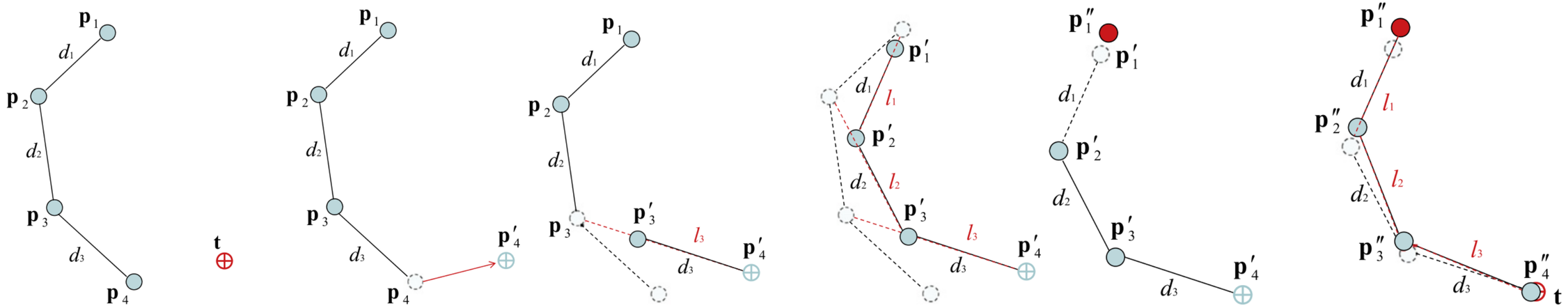


Fabrik

Iterate between

- Forward direction: Match the end-effector target
  propagate changes toward previous position to match bones' length.
- Backward direction: Match the starting position
  propagate changes toward following positions to match bones' length.

A fast iterative solver for the Inverse Kinematics problem. A. Aristidou. Graphical Models 2011.



Inverse Kinematics

Example

Synthesizing and controling skeleton animation

Blending skeleton animation

Pre-store several looping animation
Blend between animation for transition

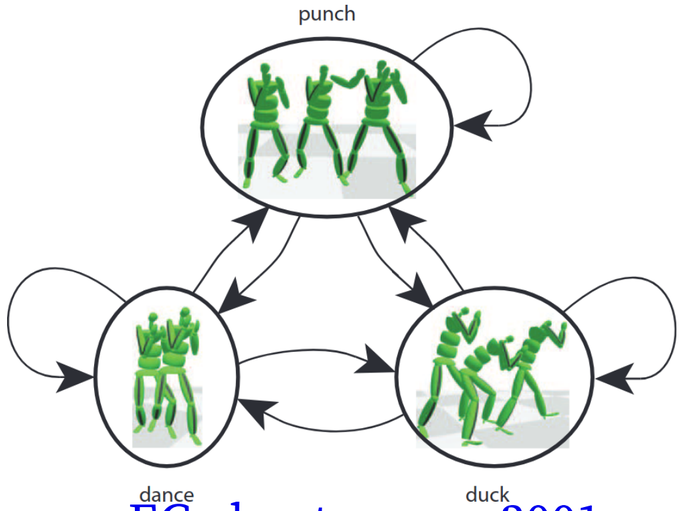
Motion graphs

Also called Move Trees (highly used in video games)

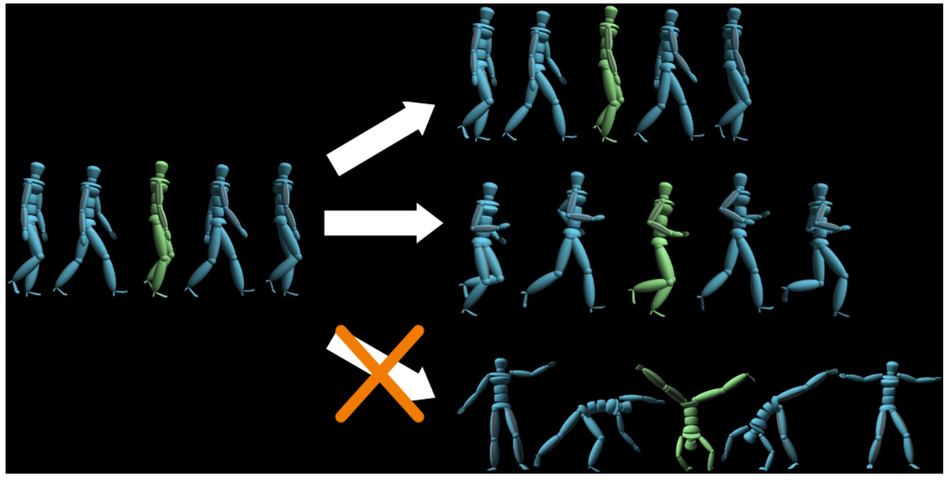
- Stores multiple precomputed animation

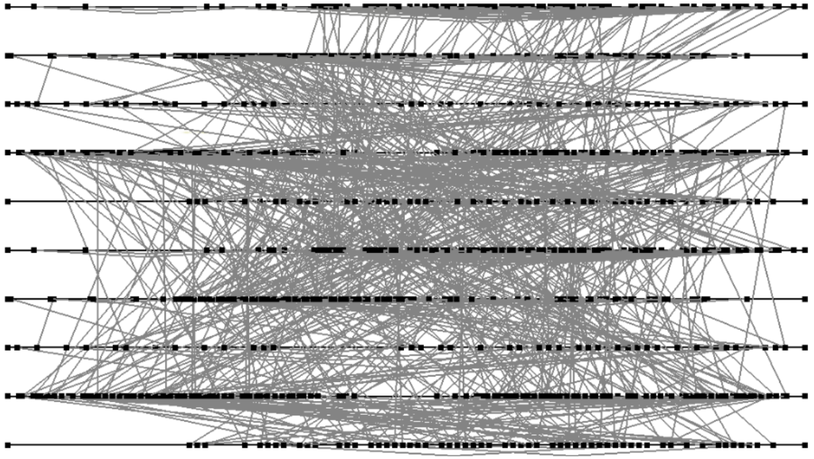
  Manually design, motion capture, etc

- Find optimal transitions between different motions



- Mizuguchi et al., Data driven motion transitions for interactive games, EG short paper, 2001
- Kovar et al., Motion Graphs, ACM SIGGRAPH 2002
- Heck and Gleicher, Parametric Motion Graphs, ACM SIGGRAPH 2007





Controlers

Mix between predefined motions and physics $\rightarrow$ allow user perturbations

1 - Define target $(\theta_{target}(t), \theta'_{target}(t))$

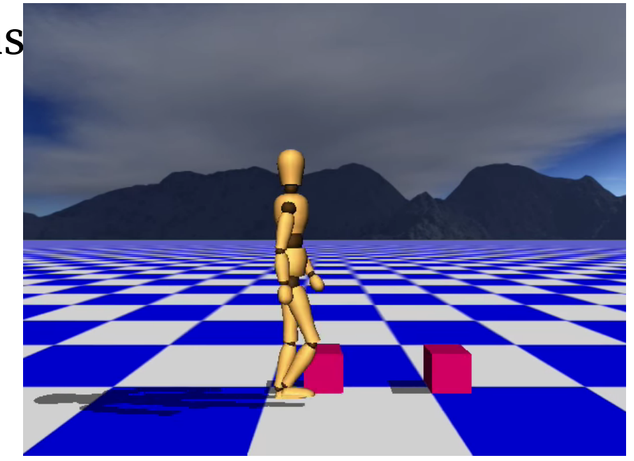
pre-defined finite state machine (Gait model)

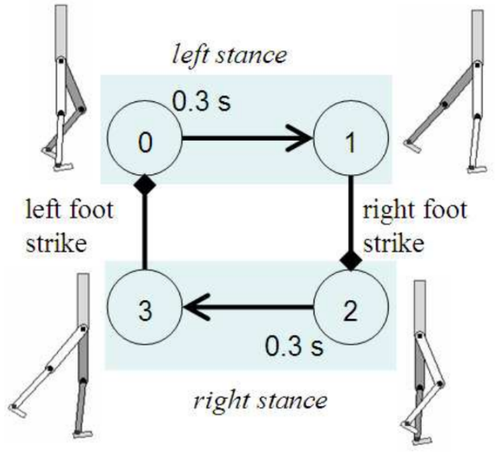
2 - Add user perturbation to the current state

3 - Use proportional derivative controlers to compute joint torque τ

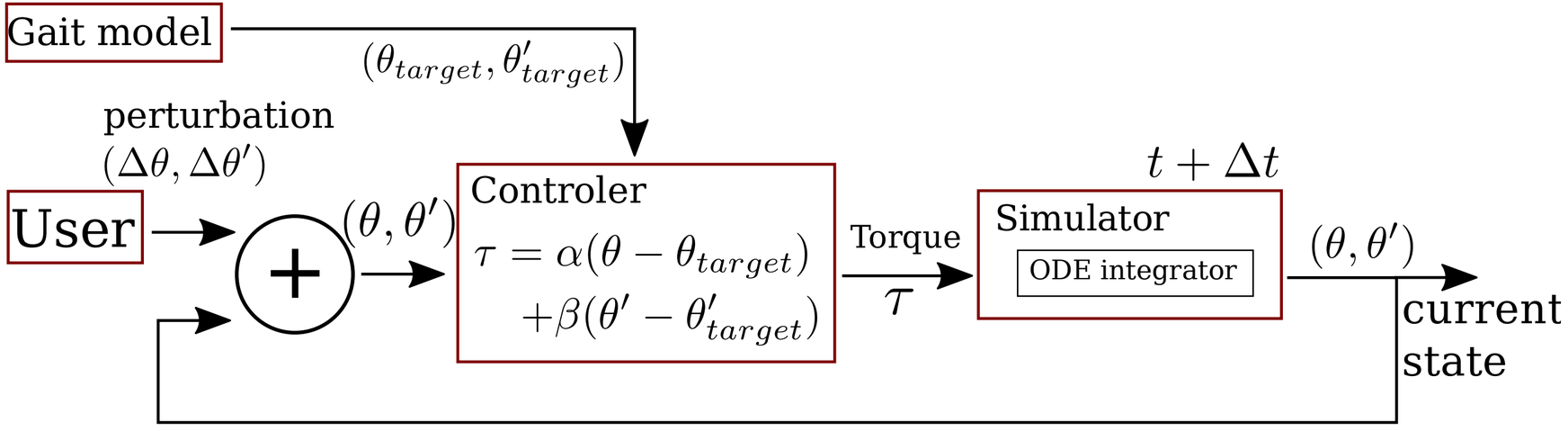
4 - Integrate torque using rigid body simulator

5 - Iterate





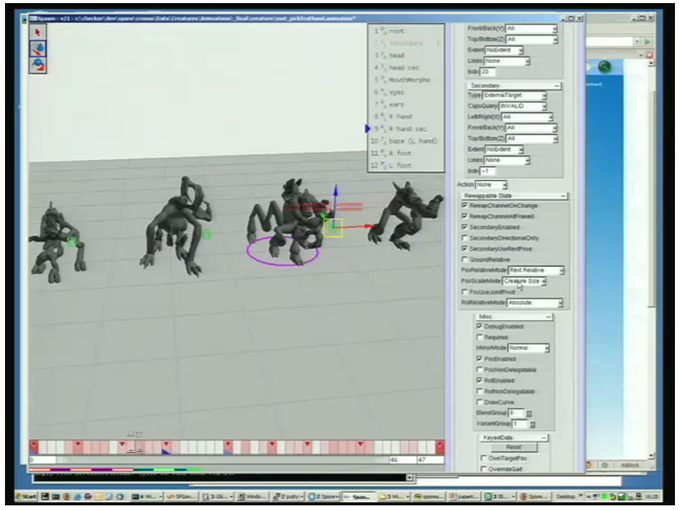
Gait model

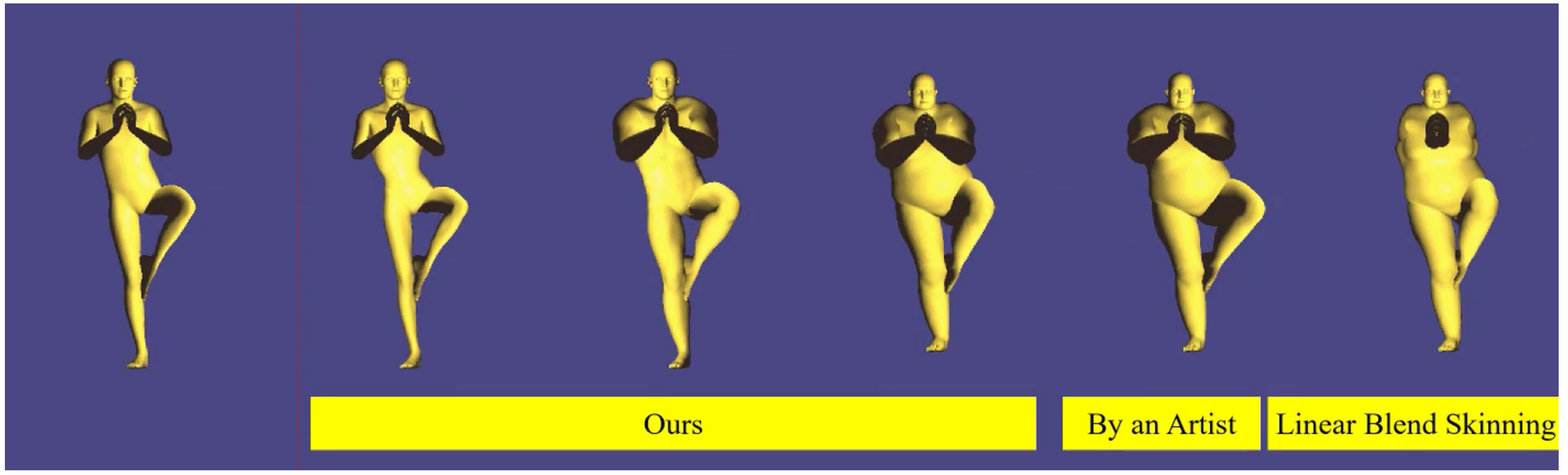


M. Raibert and J. Hodgins. Animation of Dynamic Legged Locomotion, ACM SIGGRAPH 2001
K. Yin et al., SIMBICON: Simple Biped Locomotion Control, ACM SIGGRAPH 2007

Motion transfert

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





Animation design

Based on the *Line of Action*

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

0:00 / 0:27

0:00 / 0:35

Automatic synthesis of skeletal animation

Seminal works

- [Evolving Virtual Creatures. Karl Sims. SIGGRAPH 1994]

- [Automated Learning of Muscle-Actuated Locomotion Through Control Abstraction. Radek Grzeszczuk and Demetri Terzopoulos. SIGGRAPH 1995]

- *Optimization toward objective function coupled with rigid bodies simulations*
- *Morphological variation from genetic algorithm*

Optimization of action space

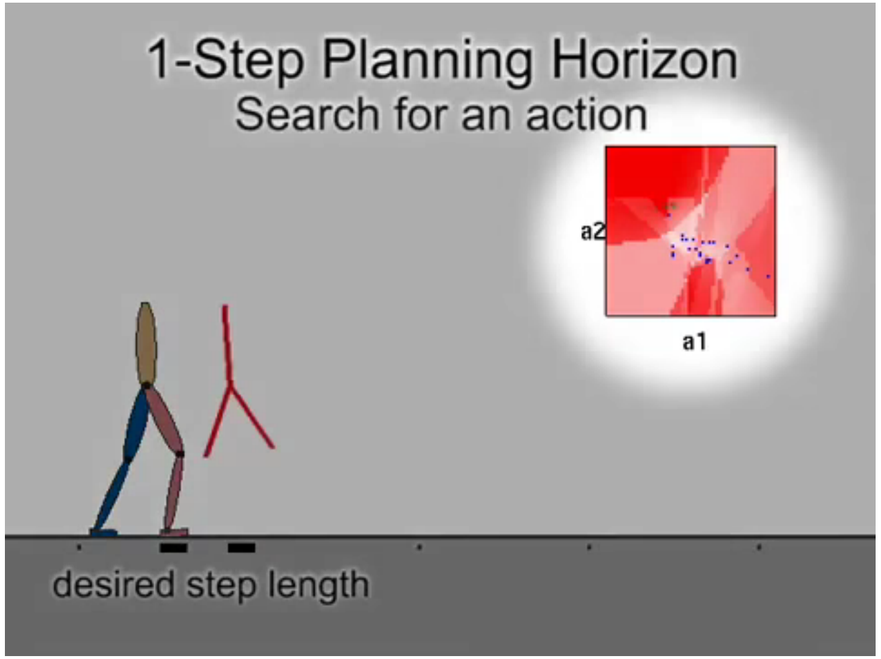
Integrating on complex terrains

- Offline simulation with random terrains
- Learn "reduced action space"
- Motion planning at run time

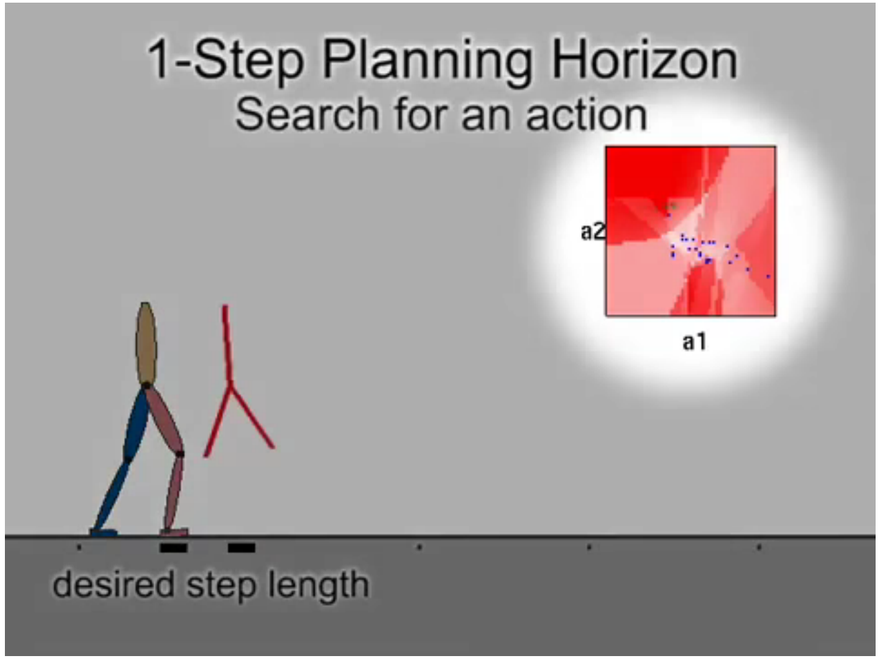
Synthesis of Constrained Walking Skills. S. Coros et al. SIGGRAPH 2008

Generate as-diverse-as-possible variation of possible motions

Diverse Motion Variations for Physics-based Character Animation. S. Agrawal et al. SCA 2013.



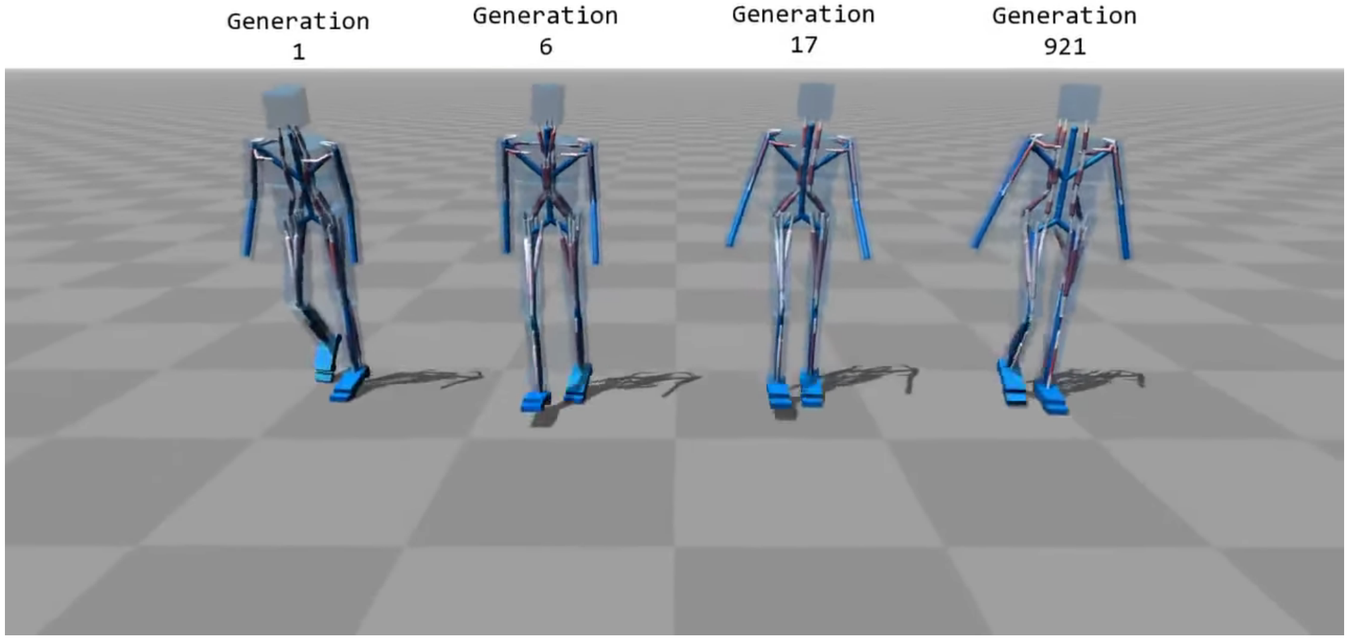
0:00 / 1:15

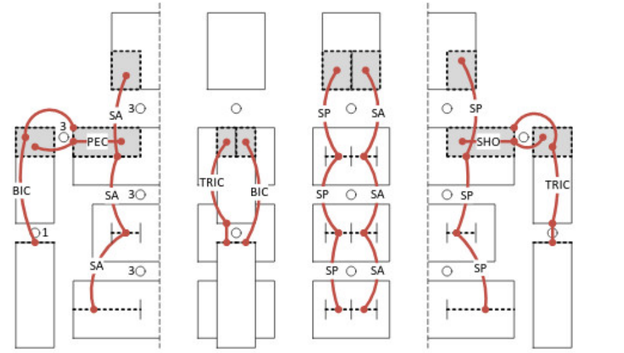

1-Step Planning Horizon
Search for an action
a2
a1
desired step length

Optimizing muscle activation

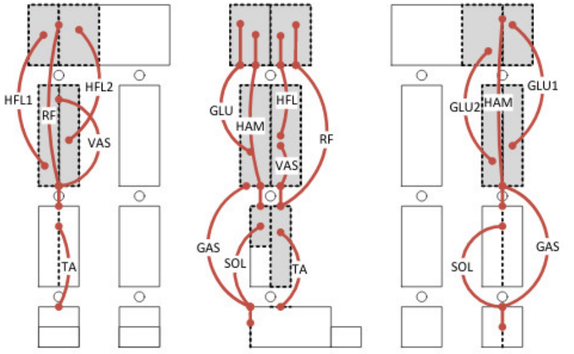
- Take into account simple biomechanical model
- Optimize sequence of activation via reinforcement learning

Flexible Muscle-Based Locomotion for Bipedal Creatures, T. Geijtenbe
2013.

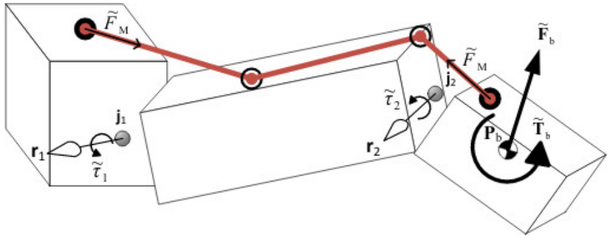




(a) Humanoid upper-body model: front, side arm, side body, and back.



(c) Humanoid lower-body model: front, side and back.

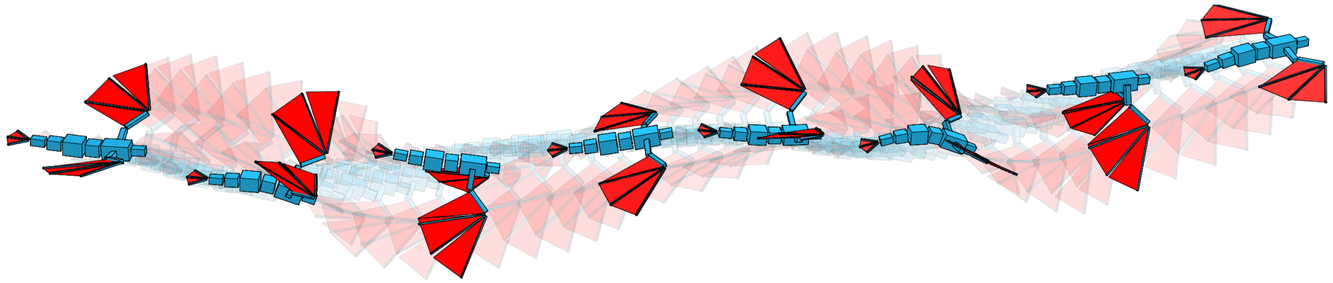


Use of deep learning

Deep reinforcement learning for complex optimization

Learn muscle activation

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



Deep learning for real-time motion control

Learn phase of the motion cycle.

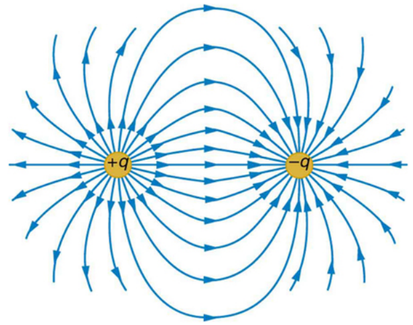
Use large data base of motion capture data

Holden et al., Phase-Functioned Neural Networks for Characte

Animating crowds of characters

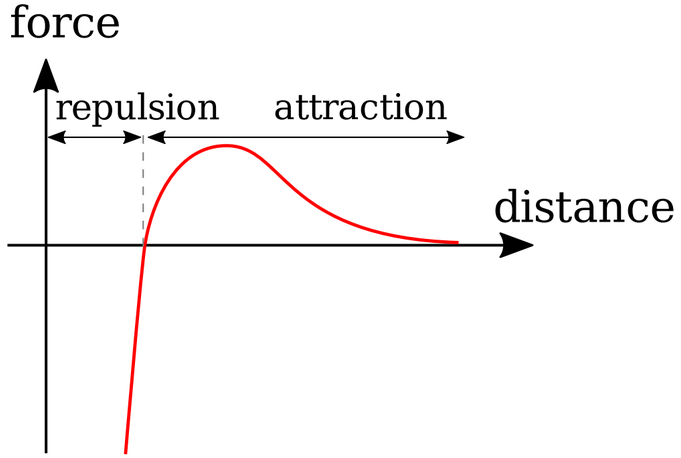
Interaction between particles

Interaction as **force field** (interact at distance)

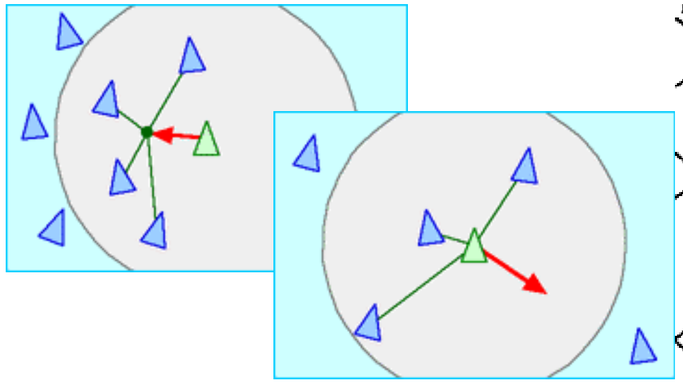


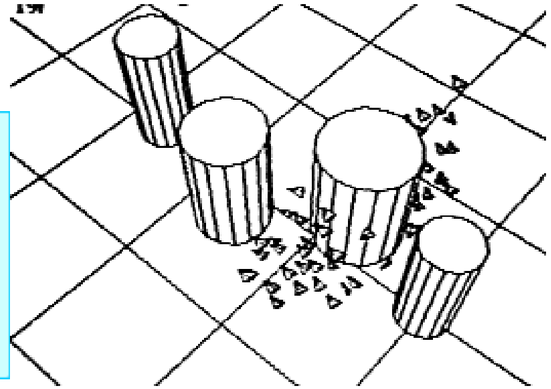
Example of usage

- Models crowd of life-like characters at large scale
  Inspired from physics particles forces (ex. Lennard-Jones potential)
  Attraction at *long-range*
  Repulsion at *short-range*



- First model: **Boids** Craig Reynolds 1987
- Extended later to human crowd modeling





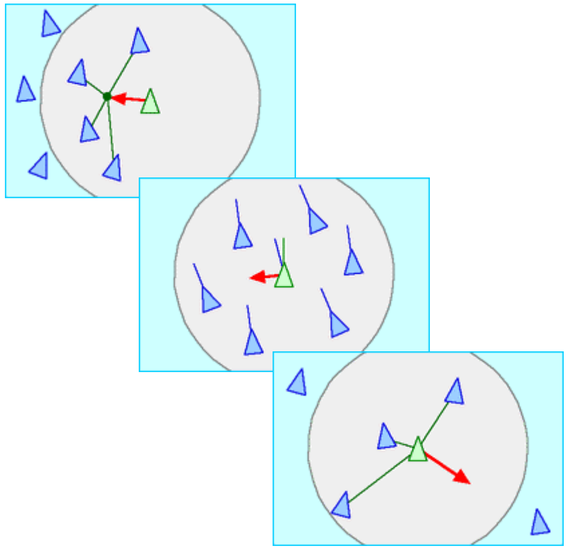
Boids Model

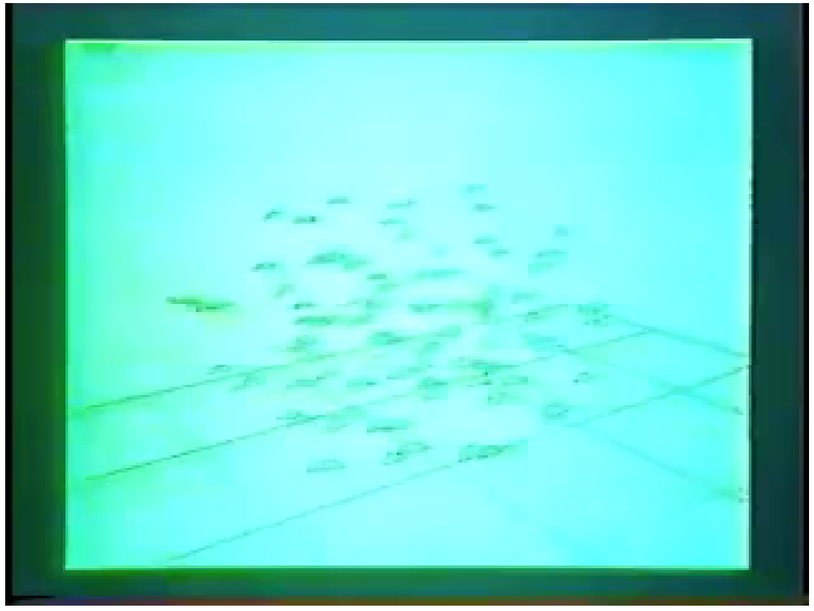
Introduced by
- [*Craig Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model, SIGGRAPH 1987*] [link]
- [*Craig Reynolds. Steering Behaviors For Autonomous Characters. Proceedings of Game Developers, 1999*] [link]

A boid is defined by its
- Position
- Speed
- Forces acting on it

Three basic local *steering behaviors* to model
- **Cohesion** between local particles
- **Alignment** between local particles
- **Separation** between too close particles

=> Leads to emerging global behaviors.





Original video in 1986 (C. Reynolds)

Boids Model - Basic model.

- Set random initial position/speed to N particles.

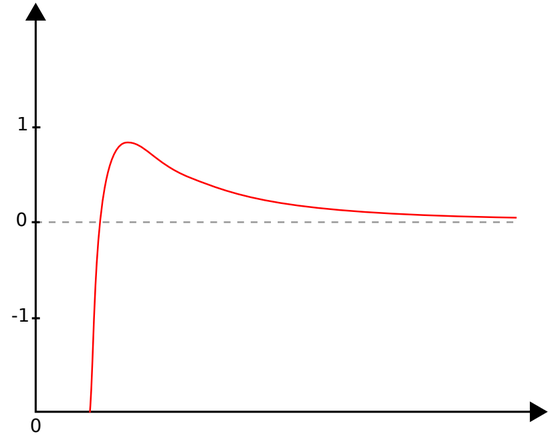
- Set attraction/repulsion force depending on pairwise distances

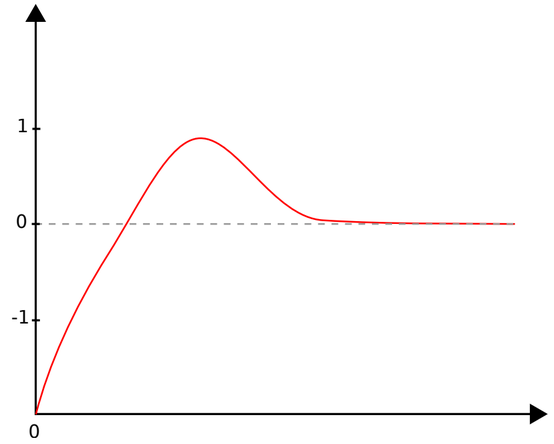
$$F(p_i) = \sum_j f(\|p_j - p_i\|) \frac{p_j - p_i}{\|p_j - p_i\|}$$

Example

- Inverse of distance $f(x) = \frac{\alpha_1}{x^2} - \frac{\alpha_2}{x^4}$

- Exponential/Gaussian $f(x) = \alpha_1 \exp\left(-k\left(\frac{x - x_0}{x_0}\right)^2\right) - \alpha_2 \exp\left(\frac{x}{x_0}\right)$





- Integrate position and speed through time

$$v^{t+\Delta t}(p_i) = v^t(p_i) + \Delta t \; F(p_i^t)$$
$$p^{t+\Delta t}(p_i) = p^t(p_i) + \Delta t \; v^{t+\Delta t}(p_i)$$

Boids Model - Complexity.

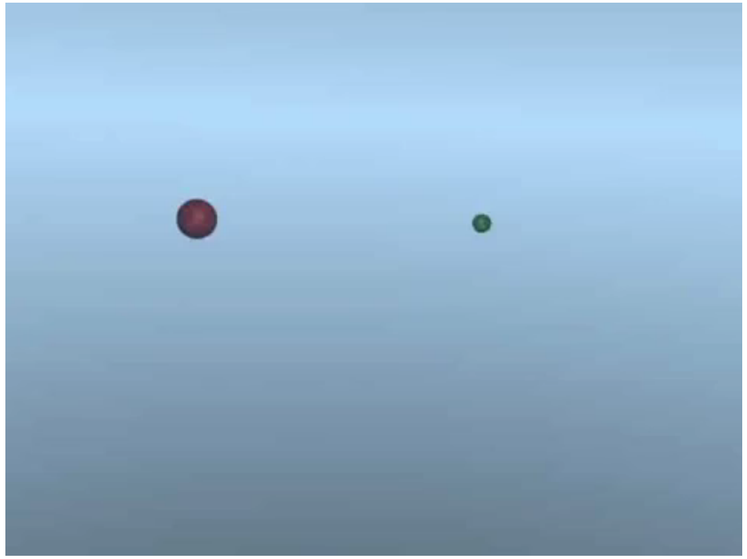
Trivial implementation:

```
struct particle { vec3 p, v, f; };

std::vector<particle> boids;

// Initialize N boids ...
// ...

// compute pairwise force
for(int i=0; i<N; ++i)
{
    for(int j=0; j<N; ++j)
    {
        if( i!=j )
        {
            const vec3& pi = boids[i].p;
            const vec3& pj = boids[j].p;

            boids[i].f += force( norm(pi,pj)) / (pi-pj)/norm(pi-pj);
        }
    }
}

// integration
for(int i=0; i<N; ++i)
{
    boids[i].v = boids[i].v + dt * boids[i].f;
    boids[i].p = boids[i].p + dt * boids[i].v;
}
```

- What is the complexity (wrt. N) of this algorithm ?
- Can you think of a way to be more efficient for large N ?

Boids Model - Usage and limitations

- Well adapted to flocks (birds, fishes - looking behavior)
- Display particles using 3D animated model



Additional behaviors

- Objective position/speed value
- Constraints: Obstacle avoidance, limited velocity
- Pursue and evade target/other particles - follow the leader, predators, etc.

- *Is collision between particles possible ?*
- *Human displacement are mostly guided by vision, what key element is missing in the basic boids force-based model ?*