COMPUTER GRAPHICS COURSE

Basic Animation

Georgios Papaioannou - 2017

FUNDAMENTAL CONCEPTS

Introduction

- Computer animation: “life” given by presenting a sequence of still images (frames) in rapid succession:
  - Sufficiently high rate → HVS perceives them as smooth motion or animation
- Minimum rate required for smooth motion ≈ 12 fps:
  - Below that, motion appears jerky
- In general, the *frames-per-second* (fps) limit is not constant; it depends on speed of movement of the objects as well as on illumination parameters

“Tweening” and Key Frames

- Traditionally, most cartoon animation was performed by tweening, the drawing of frames *in-between* key-frames
- A key frame is a characteristic (or key) snapshot of the animation sequence that contains a significant stance of the moving geometry or imagery

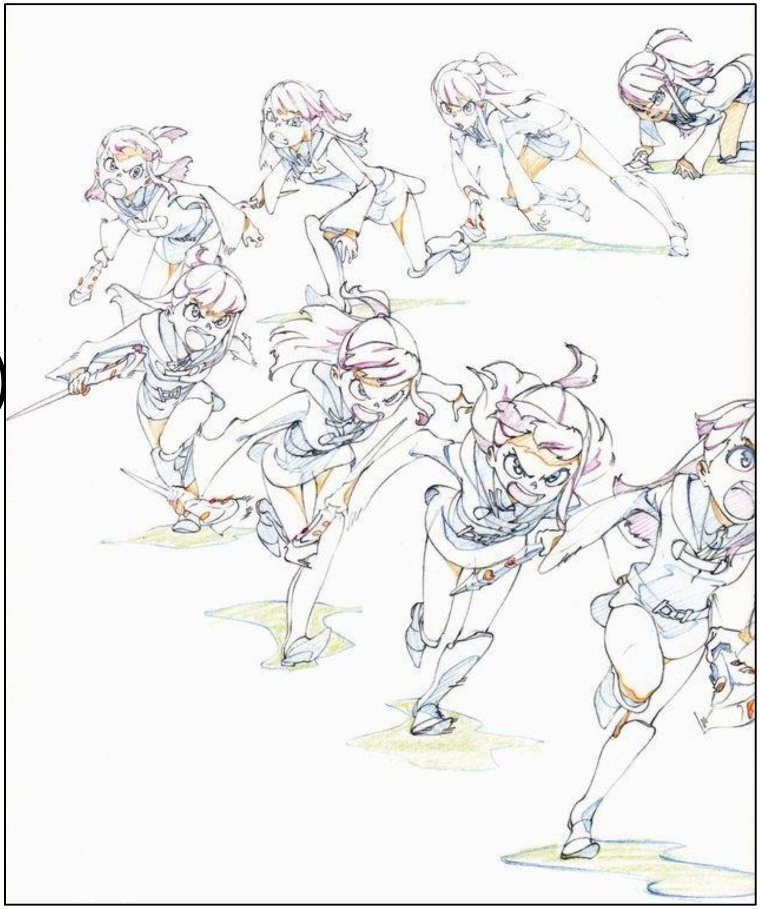
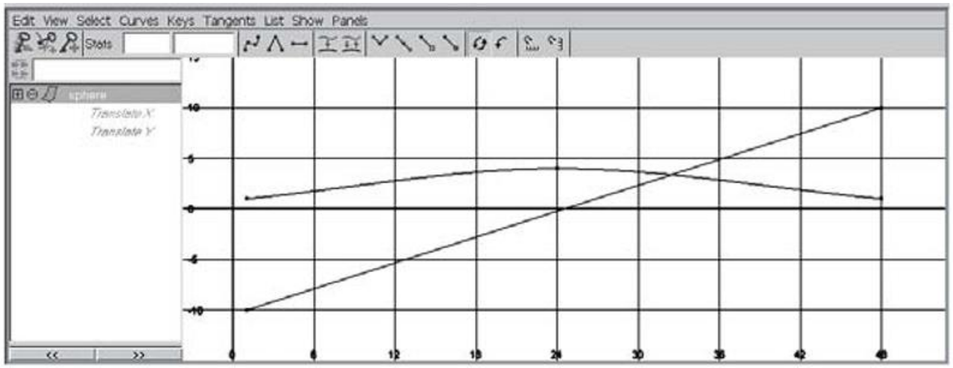
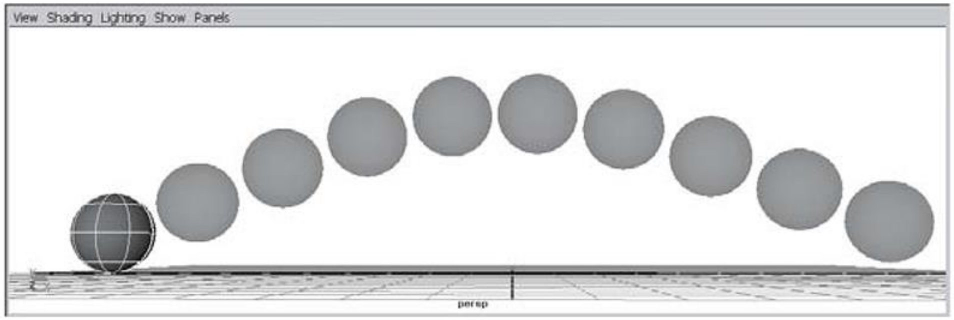


Image source: http://rebloggy.com/post/animation-little-witch-academia-little-witch-academy-studio-trigger/47083677471

Computer-based Key Frame Animation

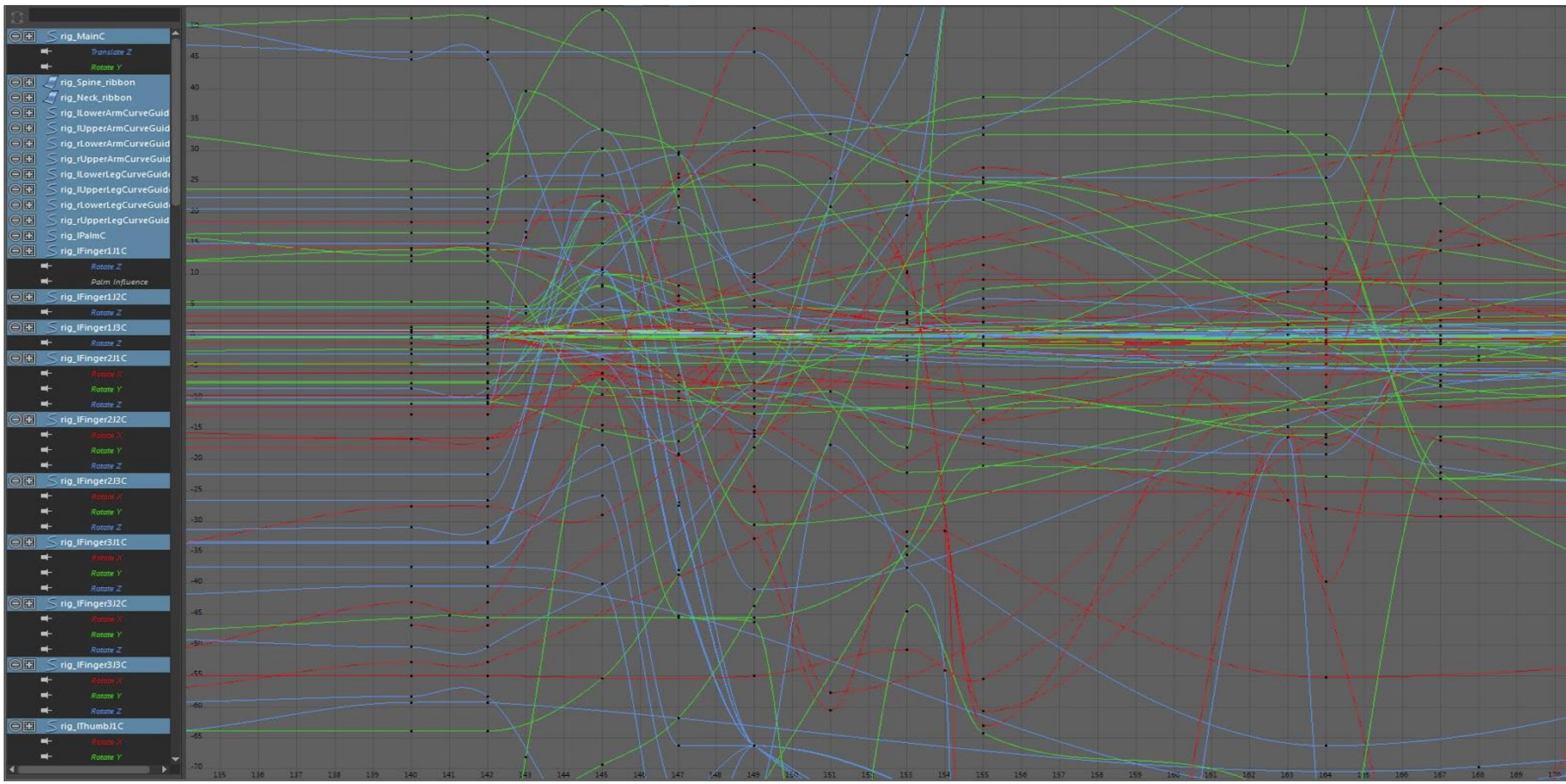
- In computer animation, key frames hold either directly the vertex data of the geometry or the parameters of higher-level entities (the simplest being transformation parameters)
- The data are interpolated using any available or suitable method (linear, spline, etc.)





Tweening and Animation Parameters

- Typical animations require a lot of parameters if tweening is performed at a low representation level



Animation Control Methods

- Impossible for an animator to define every animation variable for every frame; animation control methods have been developed
- Examples:
  - Hierarchical mesh deformation techniques
  - Procedural and representational methods for animating rigid bodies
  - Physics and collision-driven approaches
  - Skeletal animation for animating human-like or animal-like characters

Animation Control Methods

- These methods use common low level techniques such as:
  - Interpolation,
  - Collision detection
  - Motion blur
- Higher level animation control methods examined:
  - Rigid body animation
  - Skeletal animation
  - Deformable models
  - Particle systems

Procedural Animation

- The encapsulation of the animation of an object in a procedure:
  - Animation sequences can automatically be generated, often in real-time
  - Particle systems: Largest subclass of procedural animation
  - Rigid body and Skeletal animation: Can also be done procedurally
  - Behavioral animation: Subclass of procedural animation where objects determine their own actions, taking into account their environment

LOW-LEVEL ANIMATION TECHNIQUES

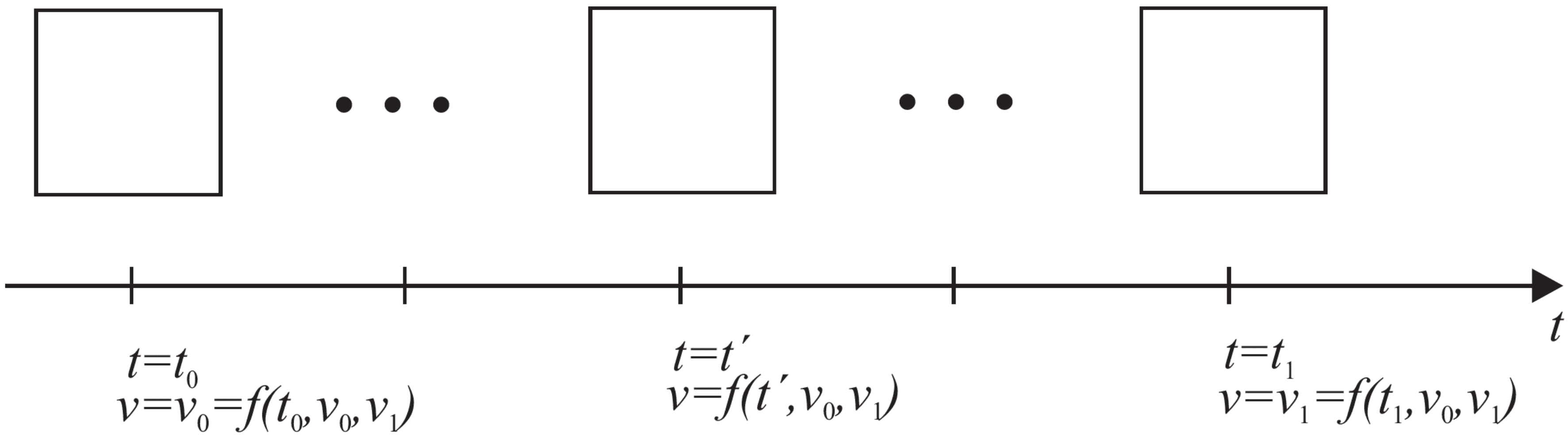
A Lower Layer of Tools

- Interpolation techniques: Means by which computer takes over the task of tweening
- Collision detection: Essential for realism by detecting when moving objects collide so that appropriate action can be taken
- Anti-aliasing in time (Motion blur): essential to most animations
- Morphing: Allows smooth transition from one graphical object to another (in a # of frames)

Interpolation

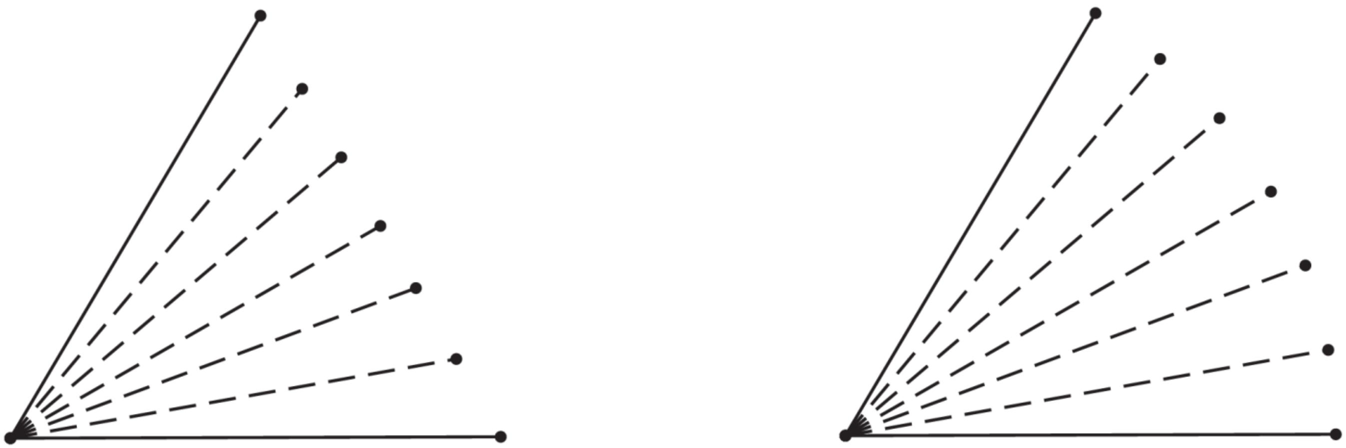
- Animation uses interpolation to do the tweening work automatically
- Extreme values of the animation variables are specified by the user
- Values of animation variables are linked to frames of the animation:
  - Since there is a 1-1 mapping between frames & time, animation variables are linked to time
- Use parametric functions $f(t)$ to interpolate the animation variables between extreme values, e.g. $\mathbf{p}(t_0)$ and $\mathbf{p}(t_1)$, which become the interpolation control points

Interpolation – Simple Example



Choosing the Right Animation Variable

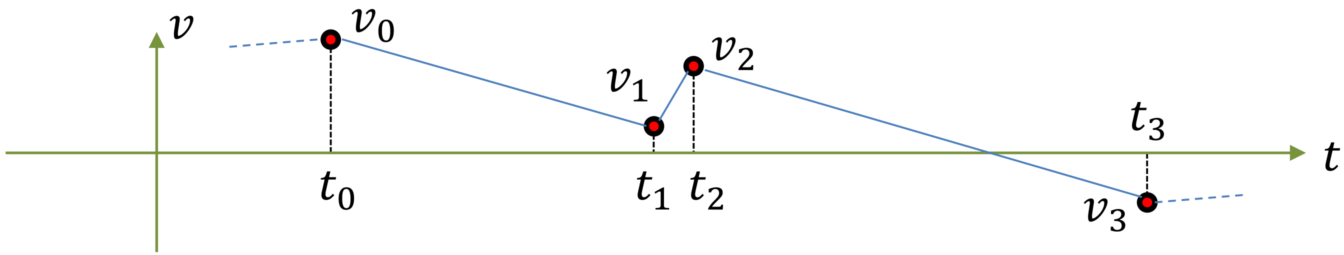
- Care must be taken in selecting the variables to be interpolated :
  - Importance of animation variable selection, e.g.: Choosing (a) the endpoint and (b) the rotation angle as animation variable



(a) (b)

Interpolation and Control Points

- Interpolation is based on a parameter t representing time
- Key frame values are control points for the animation
- They form time-value pairs, e.g.: (t_0, v_0), (t_1, v_1)
- Control points are not in general equidistant in the time domain:



- Interpolation functions pass through the interpolation control points, so $f(t_0) = v_0, f(t_1) = v_1$ for some t_0 , t_1

Linear Interpolation

- Simplest form of interpolation
- For two key points (t_0, v_0), (t_1, v_1):

$$v = v_0(1-s) + v_1 s, \quad s = \frac{t - t_0}{t_1 - t_0}$$

- For larger key point sequences, t_0 and t_1 are replaced with the nearest time stamps t_{prev} and t_{next} that t falls between:

$$t_{prev} \leq t < t_{next}$$

Linear Interpolation

- Good for:
  - Interpolating any value in dense key frame sets
  - Linear motions
  - Linear state transitions of animation control variables (e.g. Euler rotation angles)
- Bad for:
  - Sparse interpolation of positional data
  - Interpolation of dramatic pose changes

Bezier Interpolation

- Quadratic Bezier function interpolates between control values v_0 and v_2 using an extra value v_1 as an *attractor*:

$$B^2(t) = (1-t)^2 v_0 + 2t(1-t)v_1 + t^2 v_2 \quad t \in [0,1]$$

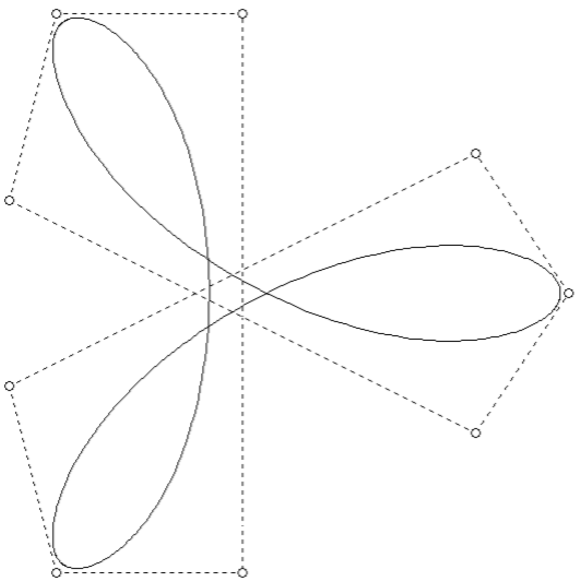
- The nth degree Bezier function interpolates between v_0 and v_n using $n-1$ attractor values v_i, i: $1 \ldots n-1$
-

Other Parametric Interp. Functions

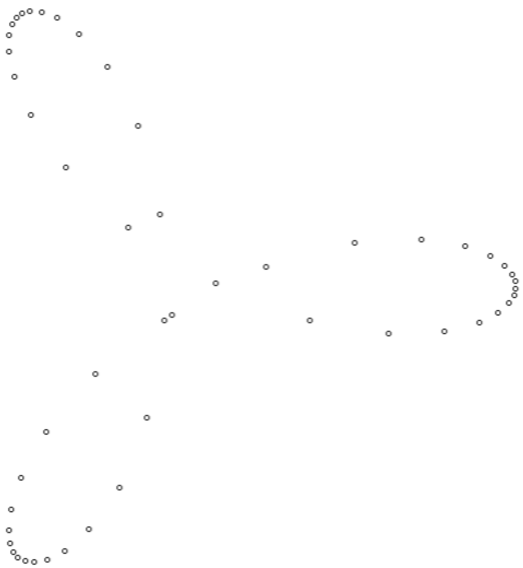
- Functions of parametric curves $\mathbf{X}(t)$ are good interpolation functions:
  - Their tangent vector $\mathbf{X}'(t)$ defines velocity → useful when used to describe motion
  - The arc length travelled along such a curve function can be computed by integrating velocity
- Caution: In general arc length travelled is not proportional to the time parameter t:
  - Can not use constant differences of t to get constant arc lengths of travel
  - If this is desired, *arc-length reparameterization* (*) of a curve is required

(*) See for example notes: http://www.personal.psu.edu/sxj937/Notes/Reparametrization.pdf

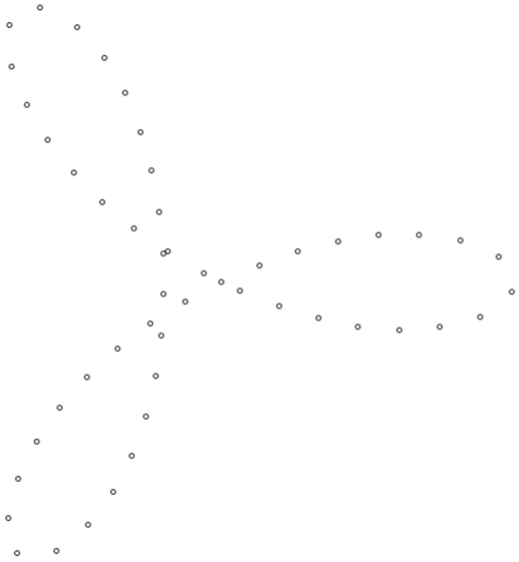
Curve Reparameterization Example



Control points



$F(t)$, equidistant t samples

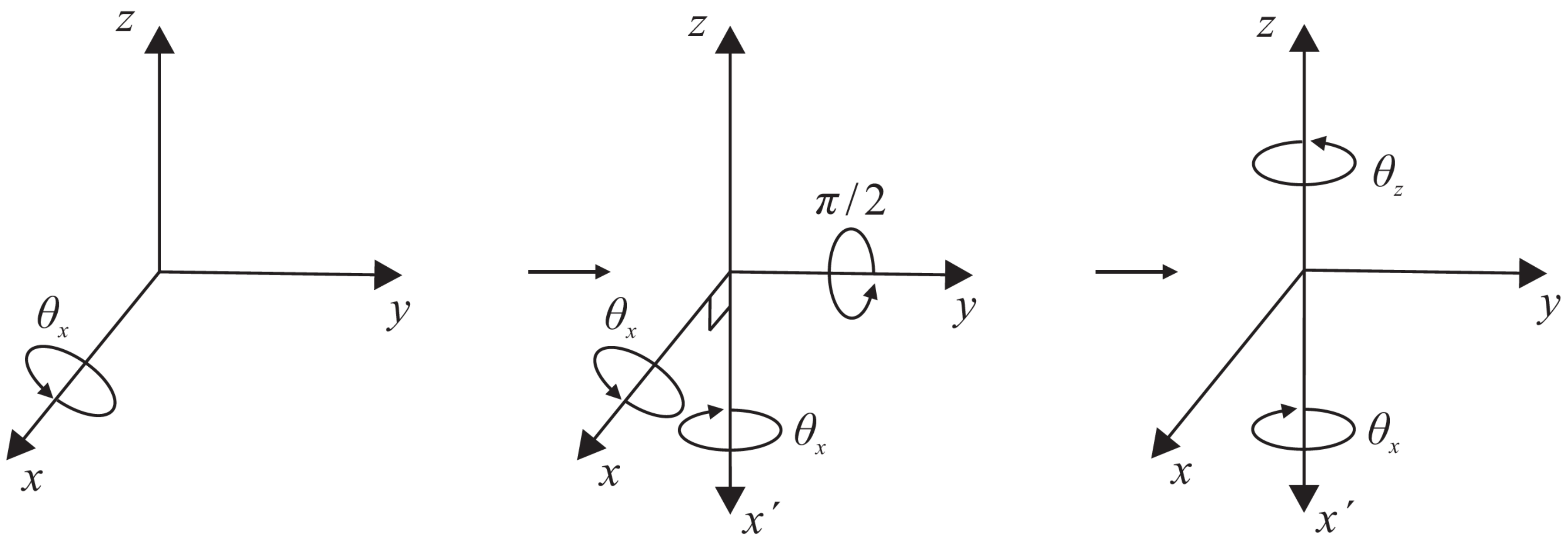


$F(s)$, equidistant s samples

Interpolating Rotations

- Suppose we express an arbitrary rotation as a synthesis of 3 basic rotations $\mathbf{R}_x(\theta_x) \rightarrow \mathbf{R}_y(\theta_y) \rightarrow \mathbf{R}_z(\theta_z)$
- Animate this by gradually incrementing θx, θy , θz → problems:
  - Rather difficult to estimate basic rotation angles that make up the required rotation about an arbitrary axis
  - Encounter a “twisting” motion, as the rotations are applied sequentially & the object seems to rotate alternately about the 3 axes
  - Encounter a phenomenon known as gimbal lock

Gimbal Lock Example



This particular sequence of rotations causes rotation around the x axis to be countered by rotation around the z axis

Quaternions

- Used as an alternative way to express rotation
- A quaternion consists of 4 real numbers: $\mathbf{q} = (s, x, y, z)$
  - s → scalar part of quaternion $\mathbf{q}$
  - $\vec{\mathbf{v}} = (x, y, z)$→ vector part of quaternion $\mathbf{q}$
- Alternative representation: $\mathbf{q} = (s, \vec{\mathbf{v}})$
- Can be viewed as an extension of complex numbers in 4D:
  - Using "imaginary units" i, j and k such that: $i^2 = j^2 = k^2 = -1$ and $ij = k, ji = -k$ and so on by cyclic permutation, quaternion q may be written as:

    $\mathbf{q} = s + xi + yj + zk$

Quaternions

- A real number u corresponds to the quaternion: $\mathbf{q} = (u, \mathbf{0})$
- An ordinary vector $\vec{\mathbf{v}}$ corresponds to the quaternion: $\mathbf{q} = (0, \vec{\mathbf{v}})$
- A point $\mathbf{p}$ corresponds to the quaternion: $\mathbf{q} = (0, \mathbf{p})$

Properties of Quaternions (1)

- Addition between quaternions:

$$q_1 + q_2 = (s_1, \vec{v}_1) + (s_2, \vec{v}_2) = (s_1 + s_2,\ \vec{v}_1 + \vec{v}_2)$$

- Multiplication between quaternions:

$$q_1 \cdot q_2 = (s_1 s_2 - \vec{\mathbf{v}}_1 \cdot \vec{\mathbf{v}}_2, s_1 \vec{\mathbf{v}}_2 + s_2 \vec{\mathbf{v}}_1 + \vec{\mathbf{v}}_1 \times \vec{\mathbf{v}}_2) =$$

$$(s_1 s_2 - x_1 x_2 - y_1 y_2 - z_1 z_2, s_1 x_2 + x_1 s_2 + y_1 z_2 - z_1 y_2,$$

$$s_1 y_2 + y_1 s_2 + z_1 x_2 - x_1 z_2, s_1 z_2 + z_1 s_2 + x_1 y_2 - y_1 x_2)$$

- Multiplication is associative
- Multiplication is not commutative

Properties of Quaternions (2)

- The conjugate quaternion of q is defined as: $\bar{q} = (s, -\vec{\mathbf{v}})$
- It holds that: $\overline{q_1 \cdot q_2} = \overline{q_2} \cdot \overline{q_1}$

- The norm of q is defined as:

$$|q|^2 = q \cdot \bar{q} = \bar{q} \cdot q = s^2 + |\vec{\mathbf{v}}|^2 = s^2 + x^2 + y^2 + z^2$$

Properties of Quaternions (3)

- It holds that: $|q_1 \cdot q_2| = |q_1| \times |q_2|$
- A unit quaternion is one whose norm: $|q| = 1$
- The inverse quaternion of q is defined as: $q^{-1} = \frac{1}{|q|^2} \bar{q}$
- It holds that: $q \cdot q^{-1} = q^{-1} \cdot q = 1$
- If $|q| = 1$ then $q^{-1} = \bar{q}$

Quaternion-based Rotations

- Quaternion rotation is more stable, requires fewer calculations & consecutive rotations can be handled in a smooth way
- Two extreme positions of the rotation can be represented by 2 unit quaternions:
- $q_0 = (1,\ \vec{\mathbf{0}})$ corresponding to the initial position and
- $\boxed{q_1 = (\sin\frac{\theta}{2}, \cos\frac{\theta}{2}\hat{\mathbf{n}})}^{*}$ corresponding to the position after rotation by θ around $\hat{\mathbf{n}}$

Applying Quaternion Rotations

- The quaternion rotation $q = (cos\frac{\theta}{2}, sin\frac{\theta}{2}\vec{\mathbf{n}}) = (s, \vec{\mathbf{v}})$ can be applied to a point $\mathbf{p}$ in quaternion form as:

$$p' = q \cdot p \cdot q^{-1} = q \cdot p \cdot \bar{q}$$

$p = (0, \mathbf{p})$

- Thus: $p' = \left(0, \underbrace{(s^2 - \vec{\mathbf{v}} \cdot \vec{\mathbf{v}})\mathbf{p} + 2\vec{\mathbf{v}}(\vec{\mathbf{v}} \cdot \vec{\mathbf{p}}) + 2s(\vec{\mathbf{v}} \times \vec{\mathbf{p}})}_{\text{Rotated point}}\right)$

Composite Rotations with Quaternions

- Expressing 2 consecutive rotations:

$$q_2 \cdot (q_1 \cdot \mathbf{p} \cdot \overline{q_1}) \cdot \overline{q_2} = (q_2 \cdot q_1) \cdot \mathbf{p} \cdot (\overline{q_1} \cdot \overline{q_2}) = (\underline{q_2 \cdot q_1}) \cdot \mathbf{p} \cdot \overline{(\underline{q_2 \cdot q_1})}$$

- The composite rotation is represented by the unit quaternion: $\boxed{q = q_2 q_1}$
- Quaternion multiplication is simpler, requires fewer operations and is numerically more stable than rotation matrix multiplication

Converting to and from Quaternions (1)

- Given a quaternion $q = (s, x, y, z)$, the corresponding rotation matrix is:

$$\mathbf{R}_q = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2sz & 2xz+2sy & 0 \\ 2xy+2sz & 1-2x^2-2z^2 & 2yz-2sx & 0 \\ 2xz-2sy & 2yz+2sx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Converting to and from Quaternions (2)

- Given a rotation matrix:

$$\mathbf{R} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The corresponding quaternion $q = (s, x, y, z)$ is:

$$s = \frac{1}{2}\sqrt{m_{00} + m_{11} + m_{22} + 1}$$

$$x = \frac{m_{21} - m_{12}}{4s}, \quad y = \frac{m_{02} - m_{20}}{4s}, \quad z = \frac{m_{10} - m_{01}}{4s}$$

Converting to and from Quaternions (3)

- If s is near zero (or zero), a different set of values for the quaternion vector part can be used:

$$x = \frac{1}{2}\sqrt{m_{00} - m_{11} - m_{22} + 1},$$

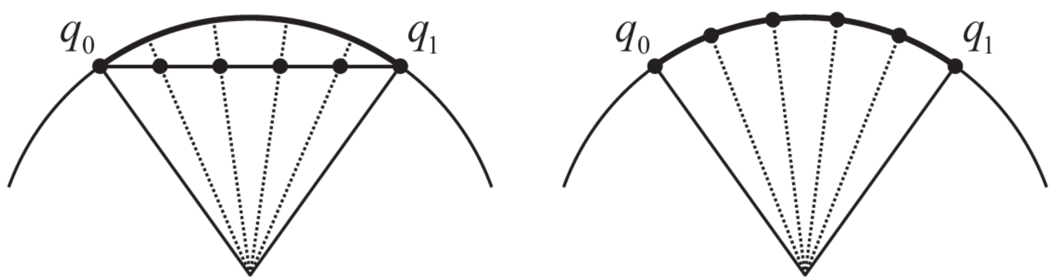
$$y = \frac{m_{01} + m_{10}}{4x}, \quad z = \frac{m_{02} + m_{20}}{4x}, \quad s = \frac{m_{21} - m_{12}}{4x}$$

Interpolating Quaternions

- Linear interpolation between these 2 quaternions will not produce expected smooth rotation between the 2 positions:
  - Instead a motion that would accelerate towards the middle
- Geometrically, unit quaternions representing rotations lie on the surface of the 4-D unit hypersphere → linear interpolation interpolates on the chord through them

Spherical Linear Interpolation

- Smooth interpolation of the rotation can be achieved by performing spherical linear interpolation (slerp):
  - Interpolation on the surface of the 4D unit hypershpere, along the great arc between $\mathbf{q}_0$ and $\mathbf{q}_1$:



$$q_S(t) = q_0 \frac{\sin(1-t)\omega}{\sin\omega} + q_1 \frac{\sin t\omega}{\sin\omega}, \quad t \in [0,1]$$

  - where $\omega = \theta/2$, θ the angle between the two directions

COLLISION DETECTION

Introduction

- At a coarse level, the objective of collision detection is to find pairs of objects which potentially intersect
- At a next level, collision detection determines:
  - if a pair of objects truly intersects
  - the intersection locus (point, edge, surface)

Introduction

- When objects collide:
  - We want to avoid inter-penetrations between solid/rigid objects
  - Respond according to Newton's third law of motion (see physics simulation)

Newton’s third law of motion:
When one body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction on the first body.

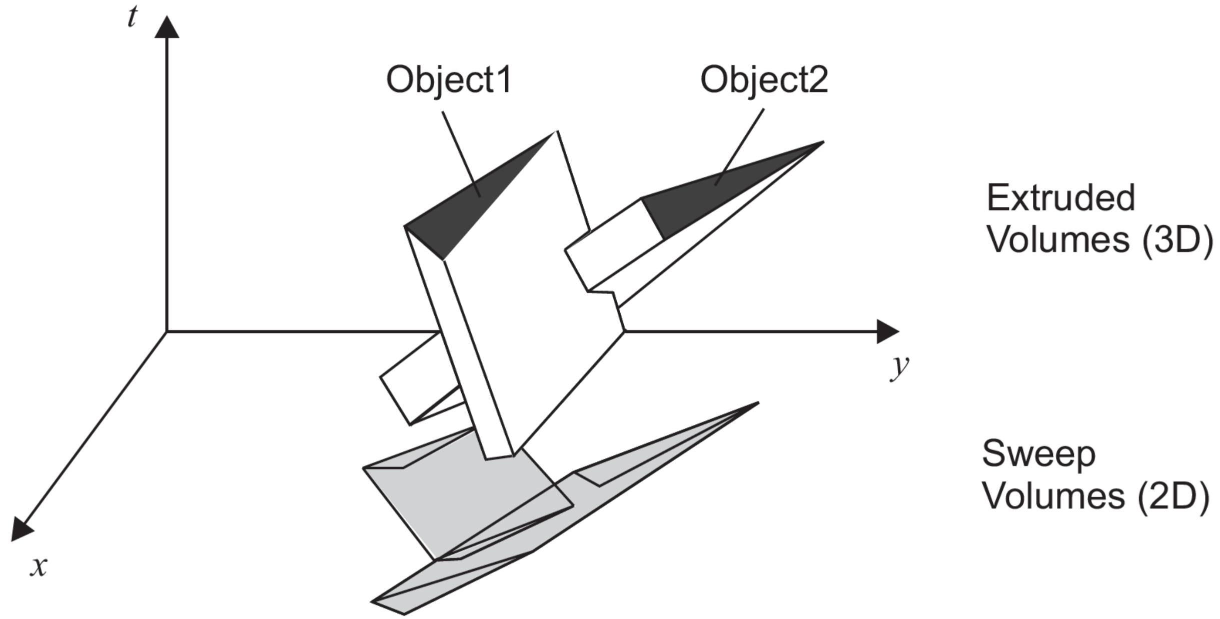
Applications

- Determine collision and hit events in 2D and 3D computer games (that subsequently cause changes in the gameplay and animation sequences)

- Determine interaction points and surfaces in physics simulation

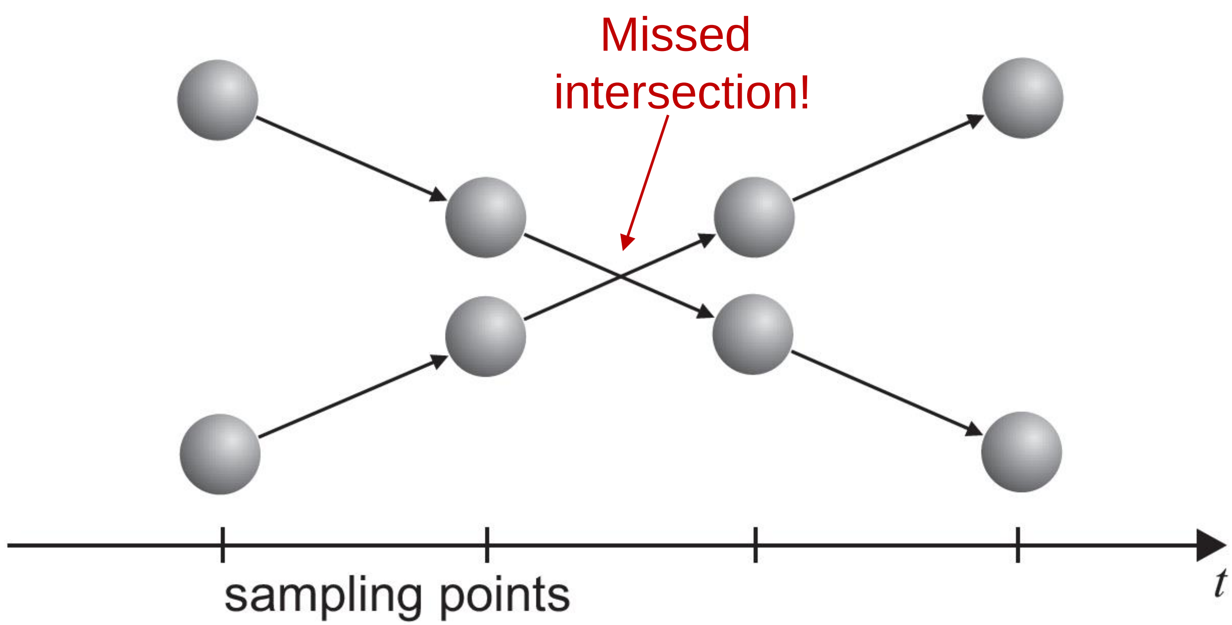
Strategies – Continuous (a Priori)

- Consider a sweep volume as the object moves (or evolves in time) between two frames
  - Accurate collision
  - Very expensive to compute, especially for non-convex objects



Strategies – Discrete (a Posteriori)

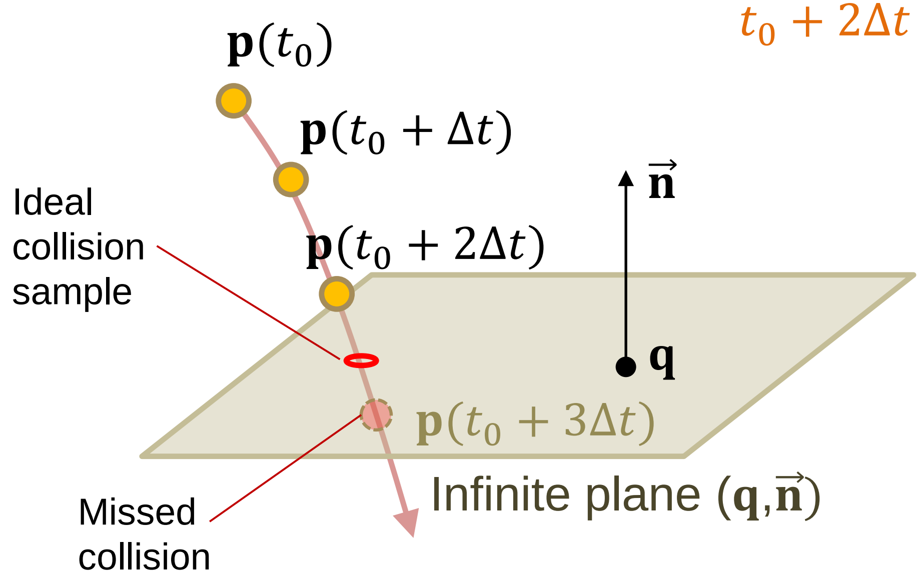
- Sample the time interval and test instances of the animated object(s) at them for intersection
  - Flexible and cheaper – favored in real-time applications
  - Can miss intersections



Improving Discrete Collisions

- Sampling interval can be refined based on intersection test results!
- Example:

Time	Test: $(\mathbf{p}-\mathbf{q})\cdot\vec{\boldsymbol{n}}$
t_0	$> e$
$t_0+\Delta t$	$> e$
$t_0+2\Delta t$	$< e$ (but not yet negative: Too close)



Indication that steps must be refined! → Adaptive sampling

Collision Detection Complexity

- In general, for N objects, $O(N^2)$ tests have to be made among them
- Each test entails $O(M_i \cdot M_j)$ primitive intersection tests, where M_i, M_j the number of primitives of the two objects
- How to improve this:
  - Limit N (collision detection is not required between all object combinations) → collision groups
  - Perform hierarchical tests
  - Make Object-object or primitive-primitive tests really cheap

Bounding Volume Hierarchies (1)

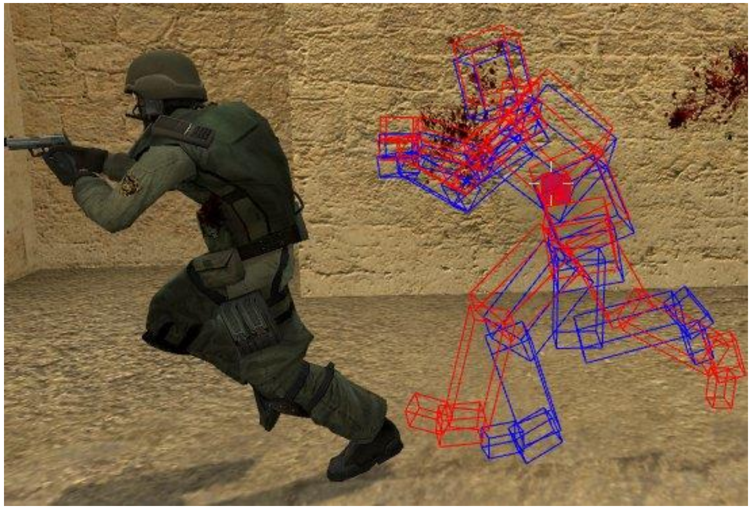
Preprocessing:

- Hierarchically group collision targets in clusters
- Find (convex) bounds for each cluster
- This bounding volume will serve as approximate collider for the cluster
- Keep grouping clusters in hierarchies

Run time, follow a branch and bound approach:

- Perform collision detection on high-level hierarchy nodes
  - If successful, proceed to check lower hierarchy levels
  - Otherwise, skip subtrees

Bounding Volume Hierarchies (2)

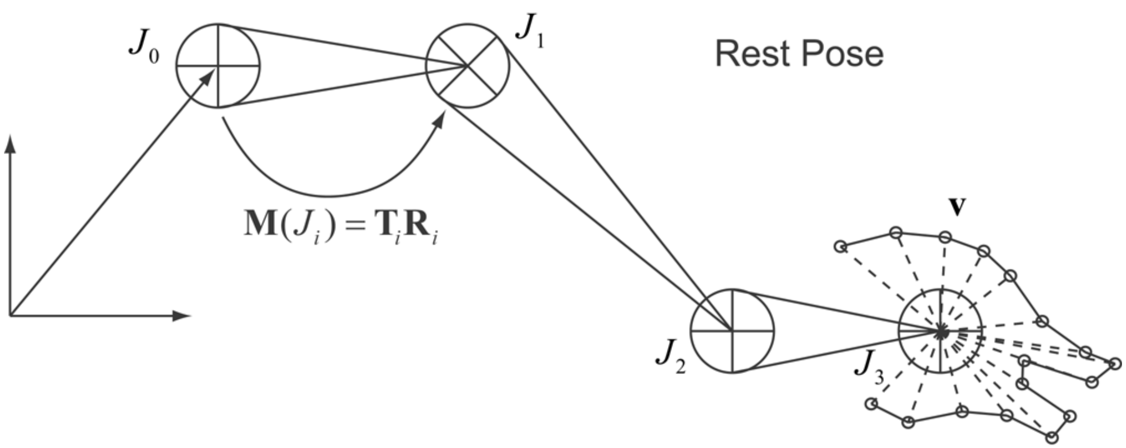


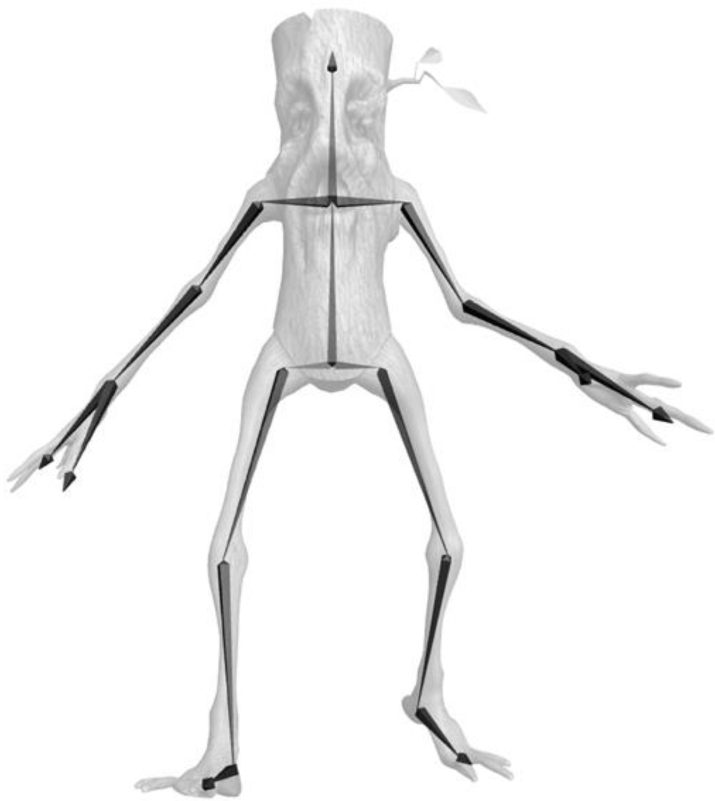
- Simple mathematical primitives such as spheres and boxes are used as bounding volumes
- Often, the bounds are manually placed or adjusted
  - Automatic bounds calculation does not know to dismiss soft or insignificant detail parts
- BVHs are also ideal for articulated models. They can follow the same structure and animation

SKELETAL ANIMATION

Skin and Bones

- We can think of a mesh as a "**skin**" that is wrapped around an animated skeleton
- The skeleton is a hierarchy of linked joints or "**bones**"





Skeletal Animation Principle

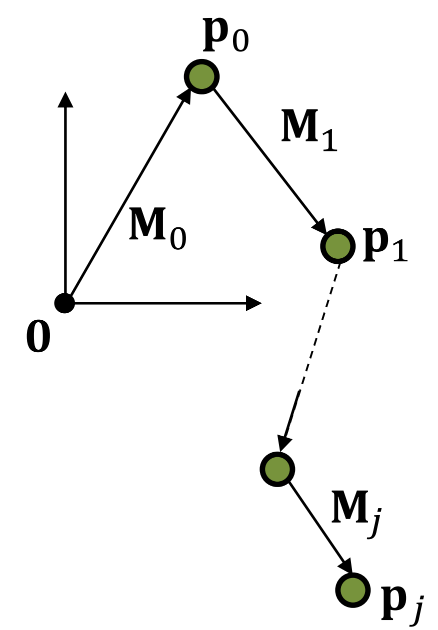
- The surface is smoothly animated by:
  - Assigning dependence weights of mesh vertices to bones (which joints the vertices "stick" to)
  - Rigidly animating the bones
  - **Interpolating the motion** of the skin from the bones

A Simple Kinematic Chain (1)

- Let us define a simple chain of dependent "joint" nodes $\mathbf{p}_j$, each one being associated with its parent via a (rigid) transformation $\mathbf{M}_j = \mathbf{T}_j\mathbf{R}_j$
- Node 0 is expressed relative to the object's (or world) reference frame, so:

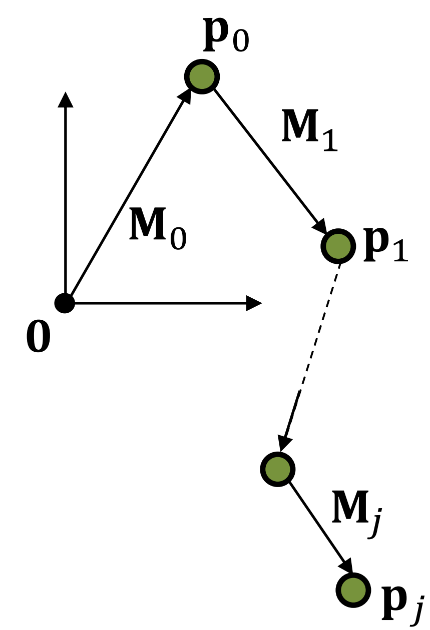
$$\mathbf{p}_0 = \mathbf{M}_0 \cdot \mathbf{0}$$

$$\mathbf{p}_j = \mathbf{M}_j \cdot \mathbf{p}_{j-1} = \underbrace{\mathbf{M}_0\mathbf{M}_1 \ldots \mathbf{M}_j}_{\mathbf{M}_{Init(j)}} \cdot \mathbf{0}$$



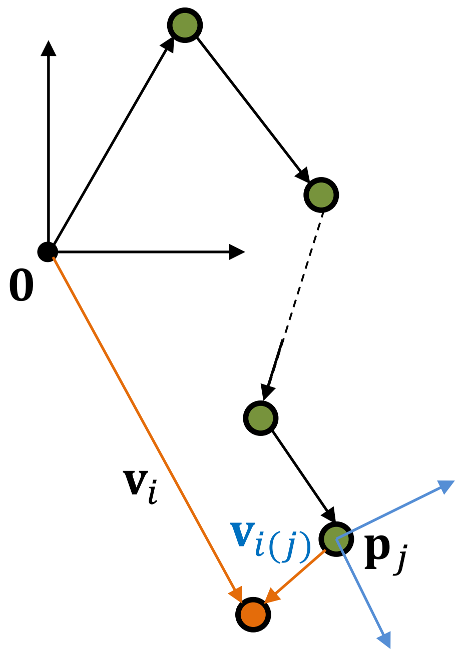
A Simple Kinematic Chain (2)

- If we initially know the positions $\mathbf{p}_j$ and no other transformation is applied, then
- $\mathbf{M}_{Init(j)} = \mathbf{T}_{\mathbf{p}_0} \cdot \mathbf{T}_{\mathbf{p}_1 - \mathbf{p}_0} \cdots \mathbf{T}_{\mathbf{p}_j - \mathbf{p}_{j-1}} = \mathbf{T}_{\mathbf{p}_j}$
- $\mathbf{M}_{Init(j)}$ expresses a point from the local reference frame of node j (centered at $\mathbf{p}_j$) to the global reference frame
- Conversely, $\mathbf{M}_{Init(j)}^{-1}$ takes an arbitrary vertex from the global coordinate system and expresses it relative to $\mathbf{p}_j$



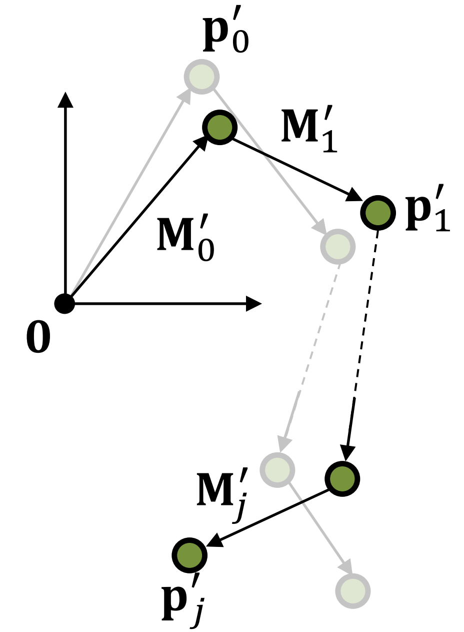
A Simple Kinematic Chain (3)

- So, given an arbitrary vertex $\mathbf{v}_i$ on the mesh to be animated, it can be expressed relative to node $\mathbf{p}_j$ using $\mathbf{M}_{Init(j)}^{-1}$:
- $\mathbf{v}_{i(j)} = \mathbf{M}_{Init(j)}^{-1} \cdot \mathbf{v}_i$



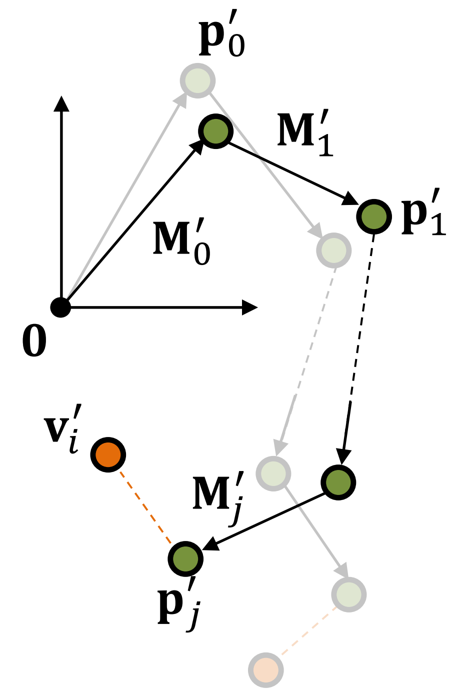
Animating the Joints

- Typically an articulated structure is animated by recalculating a new local rigid motion $\mathbf{M}'_j$ at each animation frame
- Then the changed position of each node and the respective local coordinate frame is defined w.r.t. the global system by the transformation: $\mathbf{M}_{Global(j)} = \mathbf{M}'_0\mathbf{M}'_1 \cdots \mathbf{M}'_j$



Animating the Skin Vertices

- To find the new position $\mathbf{v}_i'$ of the i-th dependent skin vertex, we simply:
  - Express it in local joint coordinates
  - Apply the new global chain transformation



$$\mathbf{v}_j' = \underbrace{\mathbf{M}_{Global(j)} \cdot \mathbf{M}_{Init(j)}^{-1}}_{\mathbf{C}_j} \cdot \mathbf{v}_j$$

- $\mathbf{C}_j$ is computed for every joint once per frame and reused for all vertices associated with the j-th joint

Soft Skinning

- If we exclusively assign each skin vertex to a joint, the resulting animated mesh will be very rigid and abrupt folds will be formed at the joints (usually with mesh self-intersections)
- We clearly want a gradual transition of the effect of each joint across the skin to result in a smooth deformation
- Solution:
  – Each vertex depends on multiple (usually adjacent) joints with a weighted contribution from each one

Vertex Weights

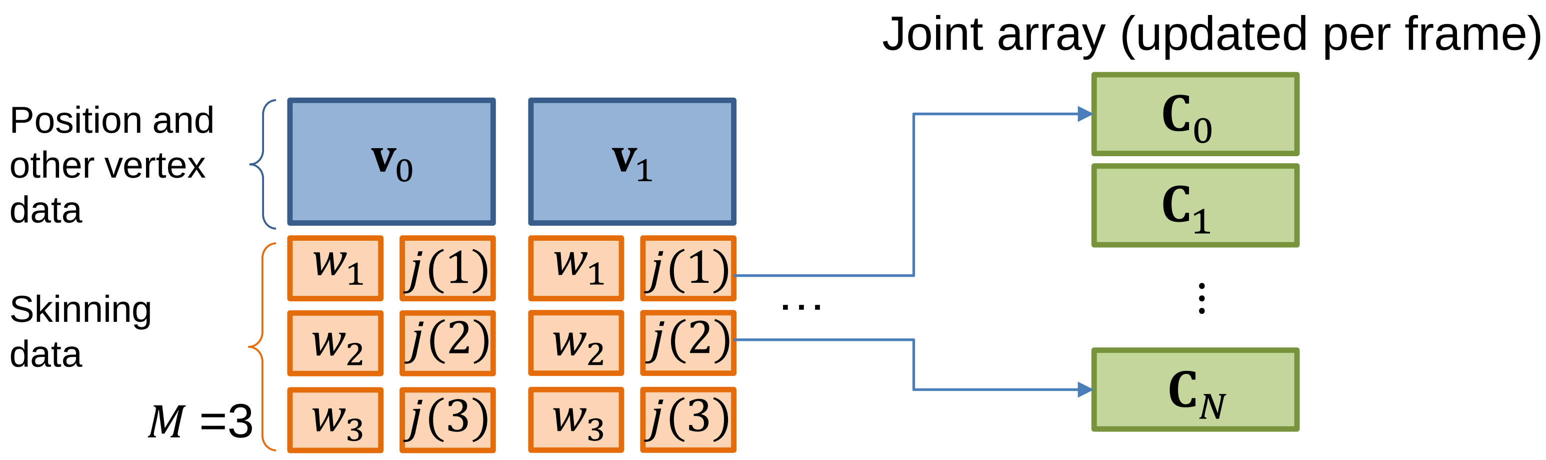
- Assume we have N joints in total and each vertex is allowed to be affected by max M of them
  - Bounding M to a small value (e.g. 4) facilitates the GPU implementation of the procedure
- Each vertex then depends on the corresponding joints using a set of (convex) weights:

$$w_k \geq 0 \quad k = 1 \dots M, \sum_{k=1}^{M} w_k = 1$$

- This in turn means that the rigid motion of a skeletal joint affects multiple vertices in a different degree of influence

Indexing the Joints

- For each weight w_k, we also need to keep track which one of the N bones is the k-th one used by vertex i
- So, each skin vertex is accompanied by:
  - A table of M precomputed weights w_k
  - A table of (integer) associations $j(k)$ between k and the ID j of the referenced joint in the (global) array of joints



Weighted Vertex Animation

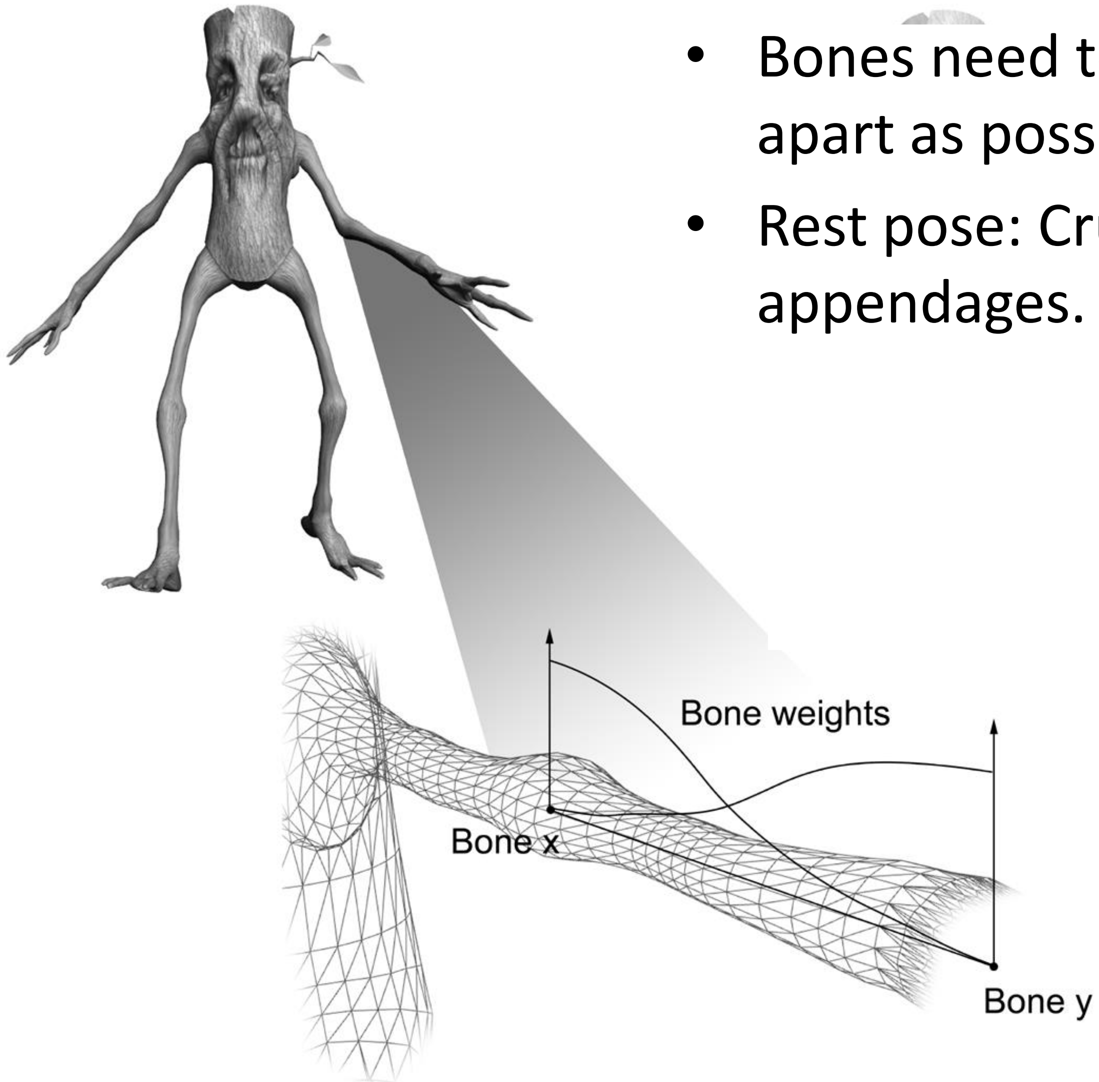
- Given now:
  - The skin vertices at rest pose $\mathbf{v}_i$
  - the joint weights w_k and indices $j(k)$,
- The final updated position $\mathbf{v}_i'$ of each skin vertex is:

$$\boxed{\mathbf{v}_i' = \left(\sum_{k=1}^{M} w_k \mathbf{C}_{j(k)}\right) \cdot \mathbf{v}_i}$$

Weight Selection

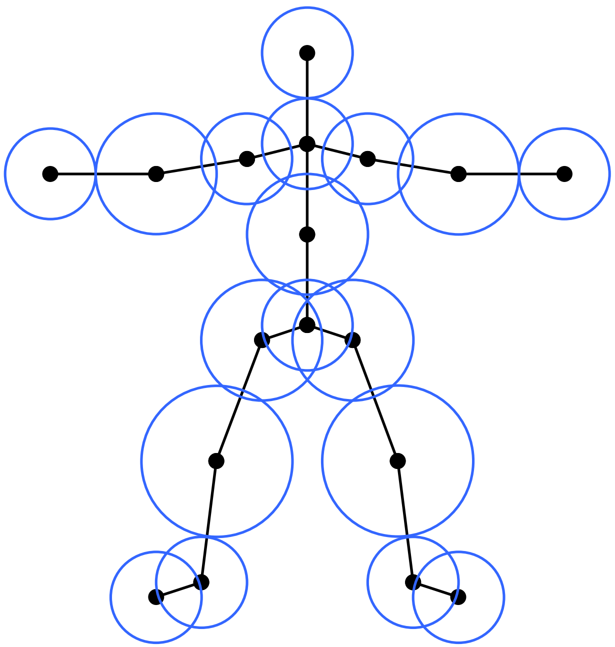
- Weights are usually assigned manually or
- Assisted by automatic weight pre-calculation:
  - Method A (Nearest neighbors)
    - Non zero weights for closest 2 bones of each vertex
    - Weighting according to distance
  - Method B (Envelope)
    - For each bone pair, assign a (unit) weight to each vertex it encounters within an area of effect (power envelop)
    - Normalize weights for each vertex (unit sum)

Weight Selection and Rest Pose (1)

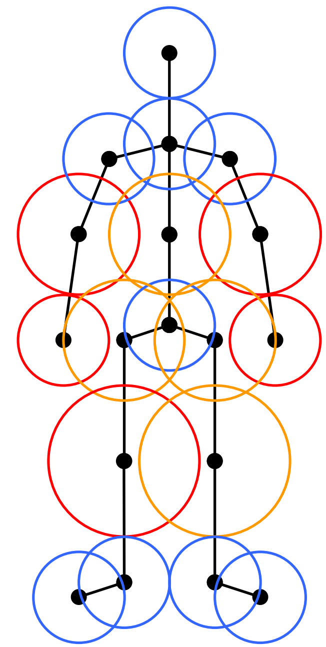


- Bones need to be initially arranged as far apart as possible (min. interference)
- Rest pose: Crucifixion with spread appendages.

Weight Selection and Rest Pose (2)



- Spread position: No interference across hierarchy branches



- Standing position: Arm and torso vertices share bones
- Same with legs

Are we Done?

- The problem with using a matrix formulation for the joint transformations is that while skinning, we weight (linearly blend) the matrices to get the desired result
- This in effect has the same problems as linearly interpolating final positions rather than orientations in key frame animation (see interpolation section)
  - Many undesirable artifacts arise, especially for large movements of the rig w.r.t the rest pose

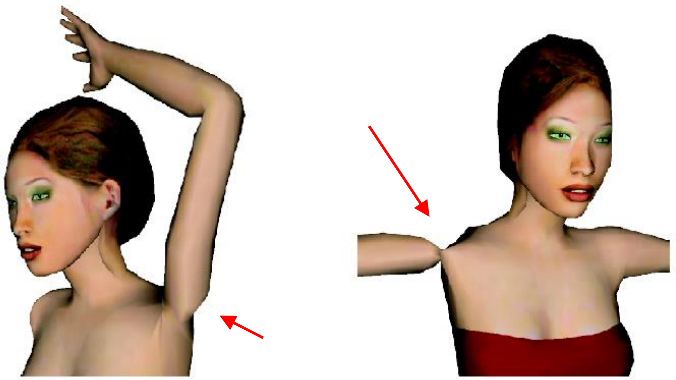


Image from [KC*07]

Dual Quaternion Skinning - Introduction

- To overcome the problem of linearly blending transformation matrices, we need to reshape the parameters to be interpolated into a more convenient form
  - This form should have similar SLERP-like interpolation (see quaternions)
  - All tuples of parameters must be compactly but fully represented (translation + rotation, scaling is not relevant)

Dual Quaternions

- Dual quaternions are an expansion of dual numbers to the quaternion form (or a quaternion of dual numbers, if you want)
- A dual number is defined similar to a complex number:
  $\hat{a} = a_0 + \varepsilon a_\varepsilon, \varepsilon^2 = 0$
- A dual quaternion is similarly defined as a quaternion of dual coordinates: $\widehat{\mathbf{q}} = \widehat{w} + i\hat{x} + j\hat{y} + k\hat{z}$
- Or the concatenation of two quaternions (follows from the above): $\widehat{\mathbf{q}} = \mathbf{q}_0 + \varepsilon \mathbf{q}_\varepsilon$

Dual Quaternions and Rigid Motion

- Multiplication of a quaternion and a dual quaternion is a dual quaternion
- A unit* (i.e. unit length) dual quaternion always represents a composition of a rotation and translation
- Given:
  - Rotation quat. of θ angle around $\mathbf{n}$: $q_0 = \left(sin\frac{\theta}{2}, cos\frac{\theta}{2}\mathbf{n}\right)$
  - Translation dual quat. by $\mathbf{t}$: $\hat{t} = 1 + \varepsilon\frac{1}{2}\mathbf{t}$
- A rotation followed by translation is:

$$\left(\hat{\mathbf{t}} = 1 + \varepsilon\frac{1}{2}\mathbf{t}\right) \cdot q_0 = q_0 + \varepsilon\frac{1}{2}tq_0, \quad t = (0, \mathbf{t})$$

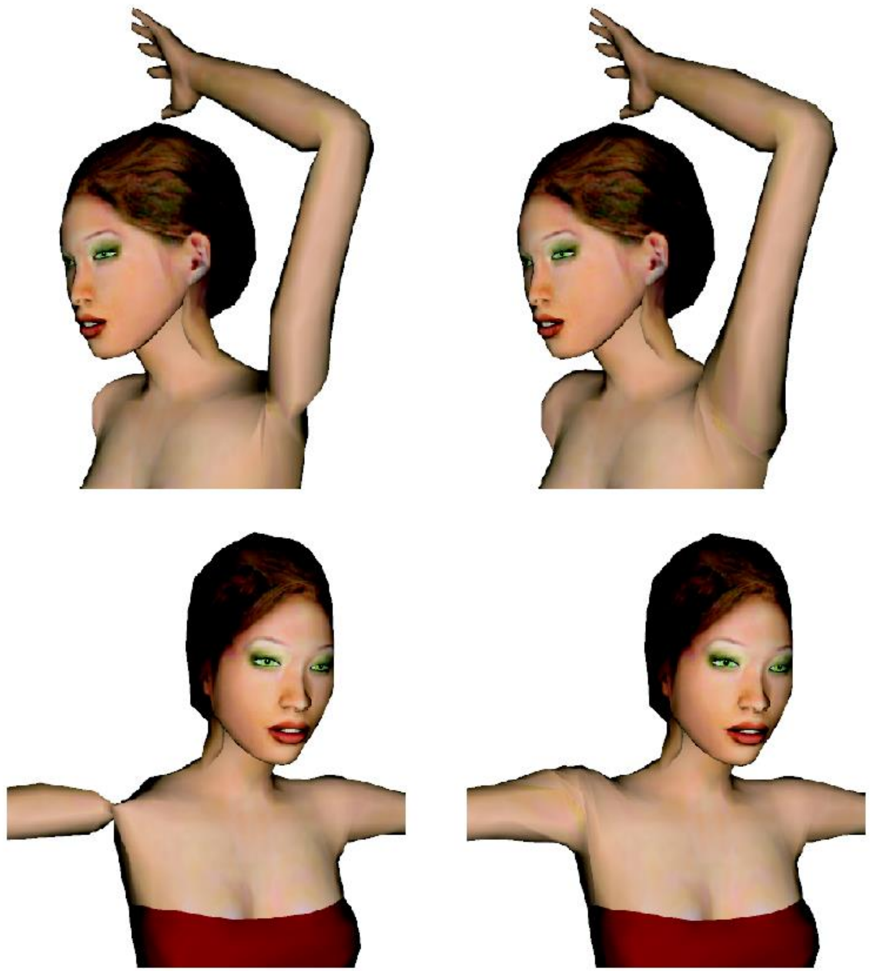
Revisiting Weighted Averages

- Having expressed all transformations in dual quaternion form,
- We can:
  - Multiply dual quaternions instead of matrices to obtain global transformations in kinematic chains →
    - Always unify dual quaternions after multiplication (to ensure that they still represent a rigid motion)
    - Greatly improves arithmetic stability
  - Perform a weighted average of dual quaternions for skinning to solve artifacts:

$$DLB(\mathbf{w}, \hat{q}_1, \dots, \hat{q}_M) = \frac{w_1\hat{q}_1 + w_2\hat{q}_2 + \cdots + w_M\hat{q}_M}{\|w_1\hat{q}_1 + w_2\hat{q}_2 + \cdots + w_M\hat{q}_M\|}$$

Dual Quaternion Results

Linear blending



Dual Quaternion
Linear blending

Motion Capture (MoCap)

It is the process of recording and transforming the motion of live subjects (humans or otherwise) in order to analyze it or transfer it to synthetic avatars

Applications of MoCap

- Film production, games and live motion transfer (interactive applications, see for example Kinect games)
- Low latency, close to real time, results can be obtained → applicable in both offline and real-time rendering apps
- Cost reduction for keyframe-based animation sequences
  - Motion complexity irrelevant
  - Animator only fine-tunes results
  - Many variations of a motion can be obtained quickly
- Complex movement and realistic physical interactions such as secondary motions, weight and exchange of forces can be easily recreated in a physically accurate manner

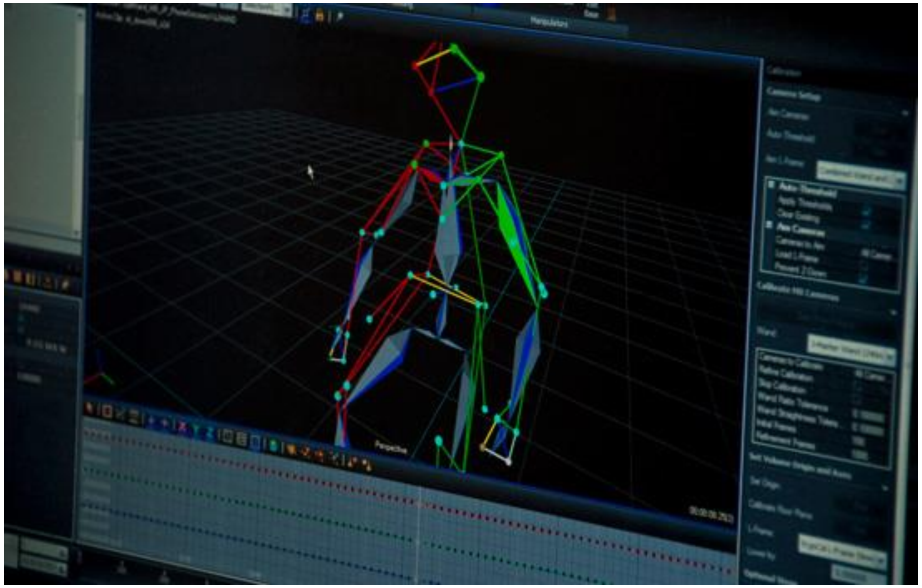
Main MoCap Technologies

- Optical Tracking
  - Marker tracking
  - Markerless tracking
  - Other
- Inertial Tracking
- Magnetic tracking

- Hybrid

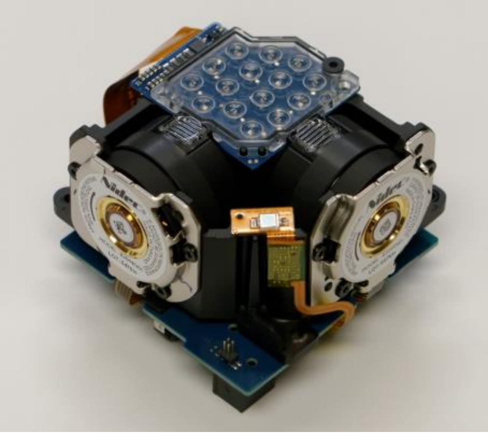
Optical Tracking

- *Optical systems* utilize data captured from image sensors to triangulate the 3D position of a point between two or more cameras calibrated to provide overlapping projections



Optical Tracking and Interaction

- Optical tracking systems are commonly used nowadays for commodity interaction devices such as:
  - Wireless position/orientation trackers (e.g. Wii mote)
  - VR headset navigation tracking (head orientation and position)



SteamVR HTC Vive Lighthouse Tracking System

Optical Tracking – Marker Tracking (1)

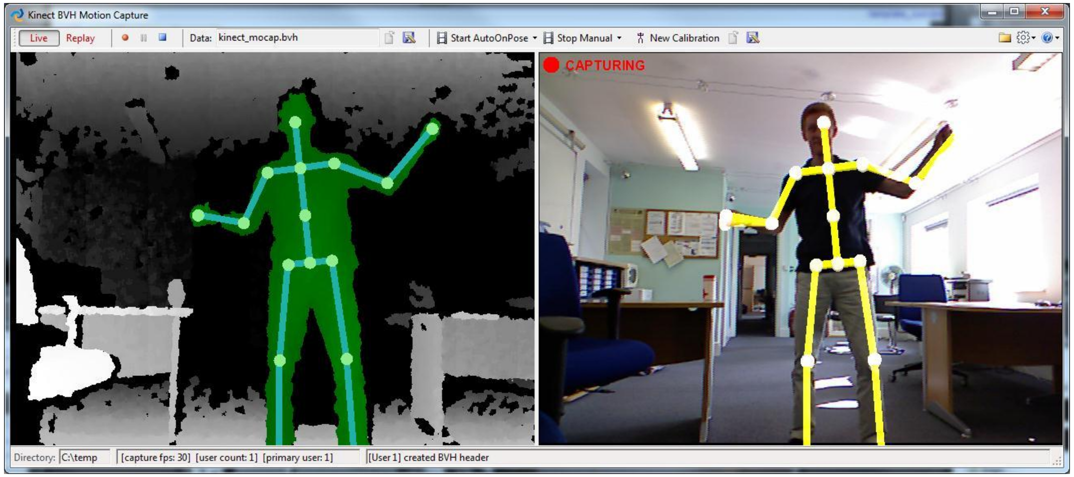
- Marker tracking. Markers (active or retro-reflective) are placed on the subject and their position reconstructed from the multiple views
- Active markers:
  - LED-powered markers enable higher marker-to-background contrast → better clarity/accuracy
  - LED activation can be time-modulated to provide unique signatures (IDs) for each marker
  - LEDs need power and control/sync circuitry

Optical Tracking – Marker Tracking (2)

- Passive markers:
  - Reflective surfaces, often beyond the visible range
  - Color coding can distinguish markers or groups/configurations of them
  - Easy to increase density
  - Very lightweight

Optical Tracking – Markerless Tracking

- Markerless tracking. Full-body 2D/3D geometry is captured (scanned) and salient points (corresponding to joints) are inferred by shape analysis
  - No obtrusive gear or suits
  - Fast setup
  - Less accurate or detailed than marker tracking
  - Ideal for interaction



Optical Tracking Properties

- Pros:
  - Lightweight user equipment
  - Easily extensible
  - Can capture groups of subjects
- Cons:
  - Occlusion issues (especially in group capture). Can be mitigated with extra sensors/cameras
  - Position capture only. Orientation must be inferred from neighboring detected points

Inertial Tracking

- Inertial measurement units (IMUs), combining a gyroscope, a magnetometer, and an accelerometer, measure rotational rates. These rotations are transmitted to a base computer translated into a skeleton

- They are handy but may loose accuracy due to error propagation and need re-sync
- Good for orientation tracking and high occlusion performances
- Often complement other tracking techniques

Magnetic Tracking

- Measure the induction current generated by the pairing of 3 perpendicular coils on both the receiver and transmitter
  - Oldest tracking system
  - Can be cumbersome
  - Immune to any non-metallic obstacles
  - Relatively small volume coverage
  - Electromagnetic interference causes distortion

TEMPORAL ANTIALIASING MOTION BLURRING

Temporal Antialiasing

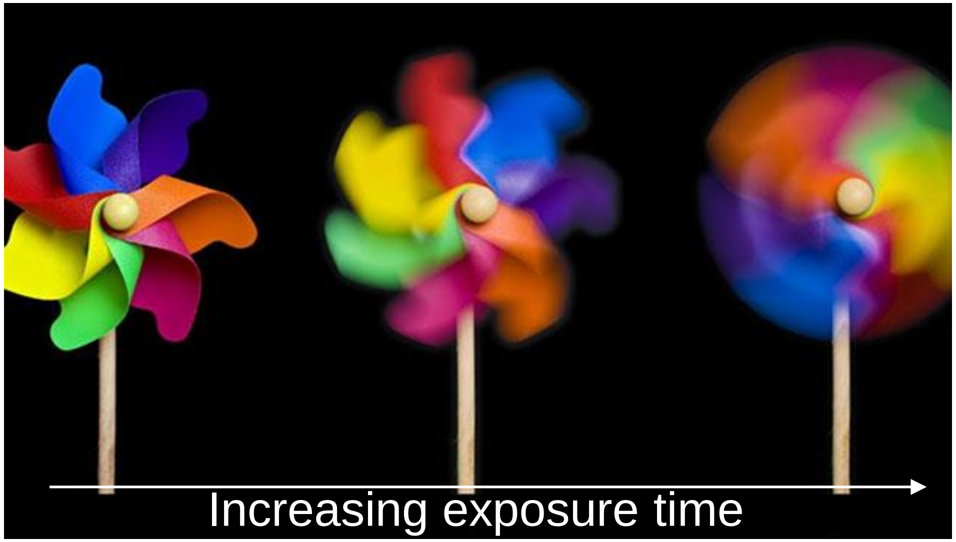
- The removal of artifacts due to high-speed animation not adequately sampled by a) the image synthesis pipeline, b) the human visual system
  - Shutter sync: A very distinctive artifact where objects appear either stationary or moving backwards due to inadequate frame rate

https://www.youtube.com/watch?v=yr3ngmRuGUc

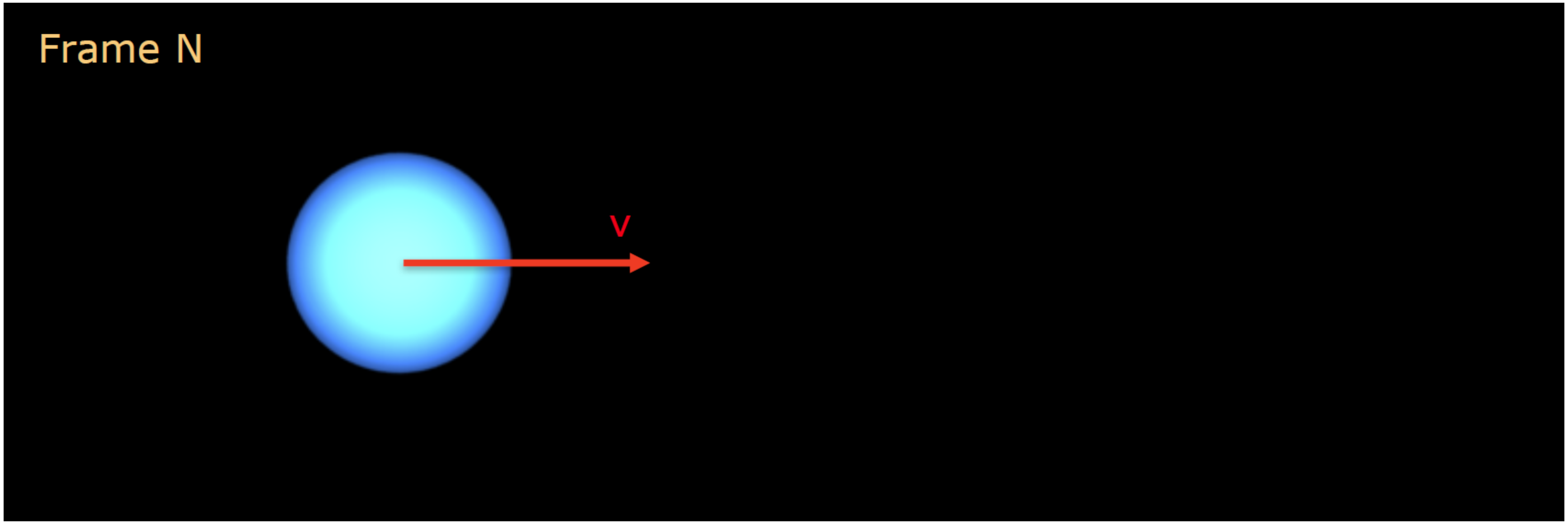
- Typically addressed with super-sampling the time domain and low-pass filtering (motion blurring)

Shutter Speed and Motion Blurring

- Motion blurring can be used to simulate the exposure time of a physical camera:
  - The sensor accumulates light during the interval the shutter is open (exposure time)
  - Corresponds to an integral of the input light over the open shutter time interval

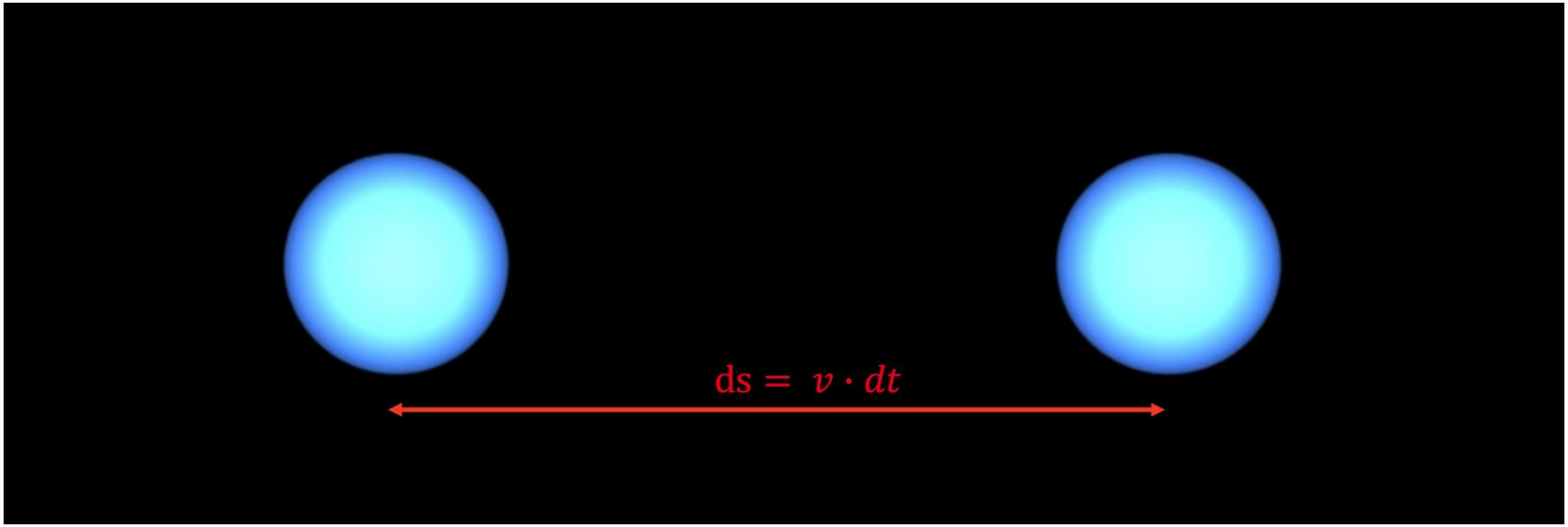


Fast-moving Objects (1)



Fast-moving Objects (2)

Fast-moving Objects (3)



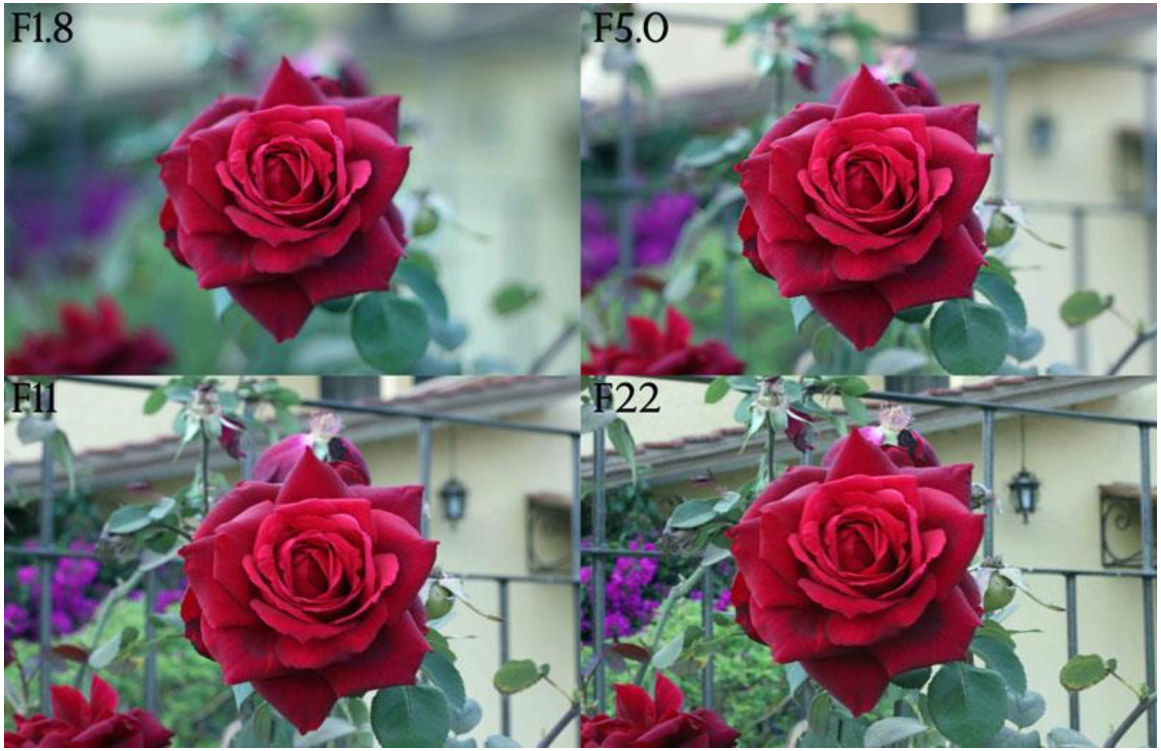
- Motion appears jerky
- The motion is under-sampled: No information is present about the in-between positions of the object →
- Aliasing in the temporal domain
- Higher velocity → greater aliasing (frame rate is fixed)

Solutions

- Increase the frame rate
- Pre-filter the signal (filter before sampling)
- Post-filter the signal (filter after sampling)

- Similar strategies to spatial antialiasing!

Camera Aperture



Camera Shutter Speed

- Controls the time interval the shutter remains open

AUEB COMPUTER GRAPHICS GROUP

Mathematical Formulation

- Measurement equation: Responsible for gathering the energy at a single pixel captured by the sensor. Accounts for:
  - Lens aperture
  - Exposure time

5D integral $(x, y, \varphi, \theta, t)$

$$L(\mathbf{x}) = \int_{t_1}^{t_2} \int_{D(\mathbf{x})} \int_{\Omega} L(\mathbf{s}, \omega, t) W(\mathbf{s}, \omega, t) d\omega d\mathbf{s} dt$$

- Considering only the temporal domain:

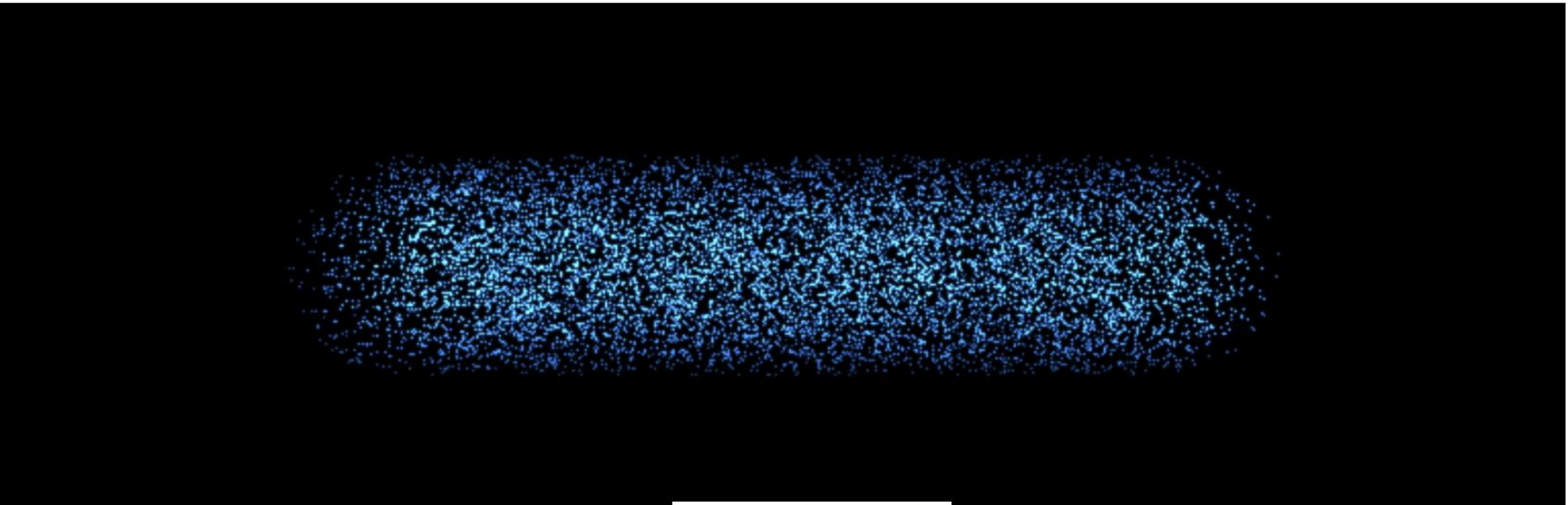
$$L(\mathbf{x}) = \int_{t_1}^{t_2} L(\mathbf{x}, t) W(\mathbf{x}, t) dt$$

zero aperture and non-zero shutter

Reconstruction filter

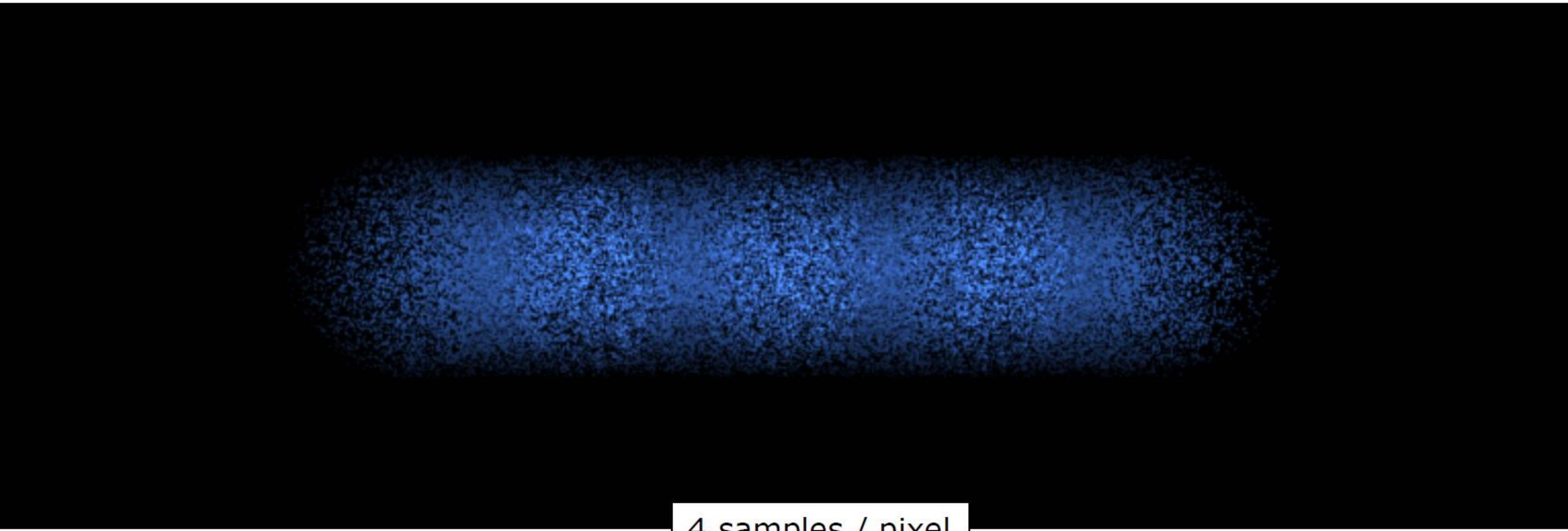
Antialiasing with Stochastic Sampling (1)

- The temporal dimension is sampled at random values of t, $t_1 \leq t \leq t_2$ and the results are weighted according to $W(\mathbf{x}, t)$



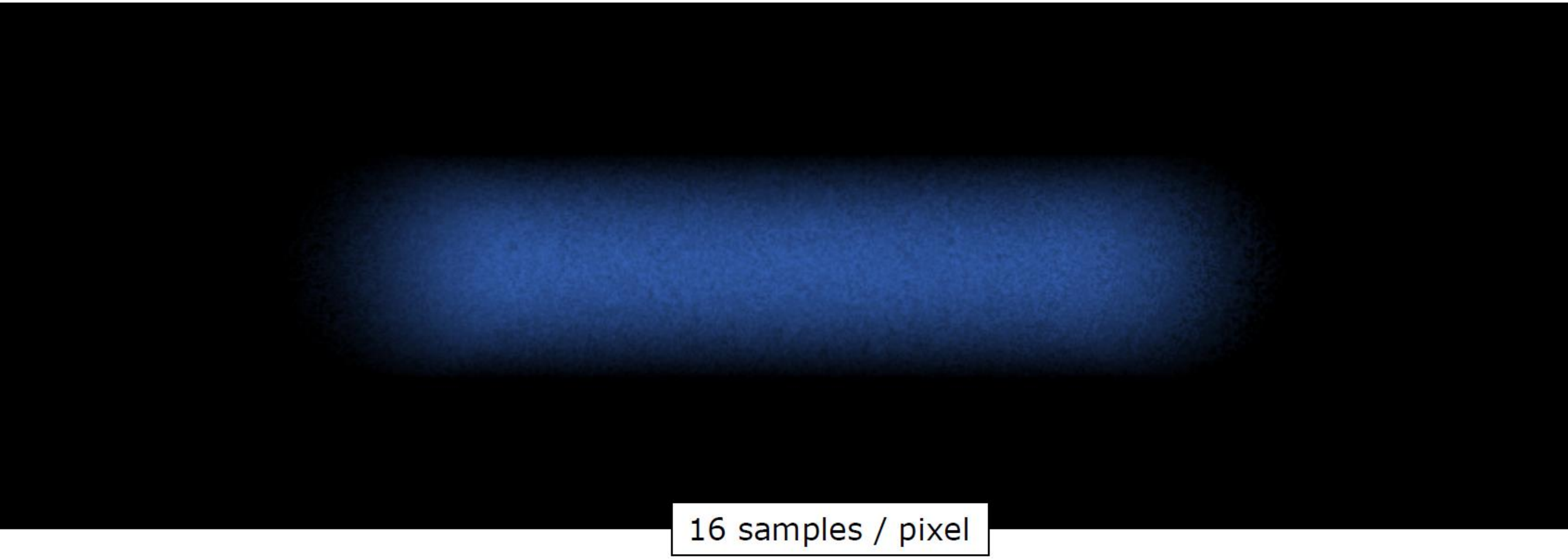
1 sample / pixel

Antialiasing with Stochastic Sampling (2)

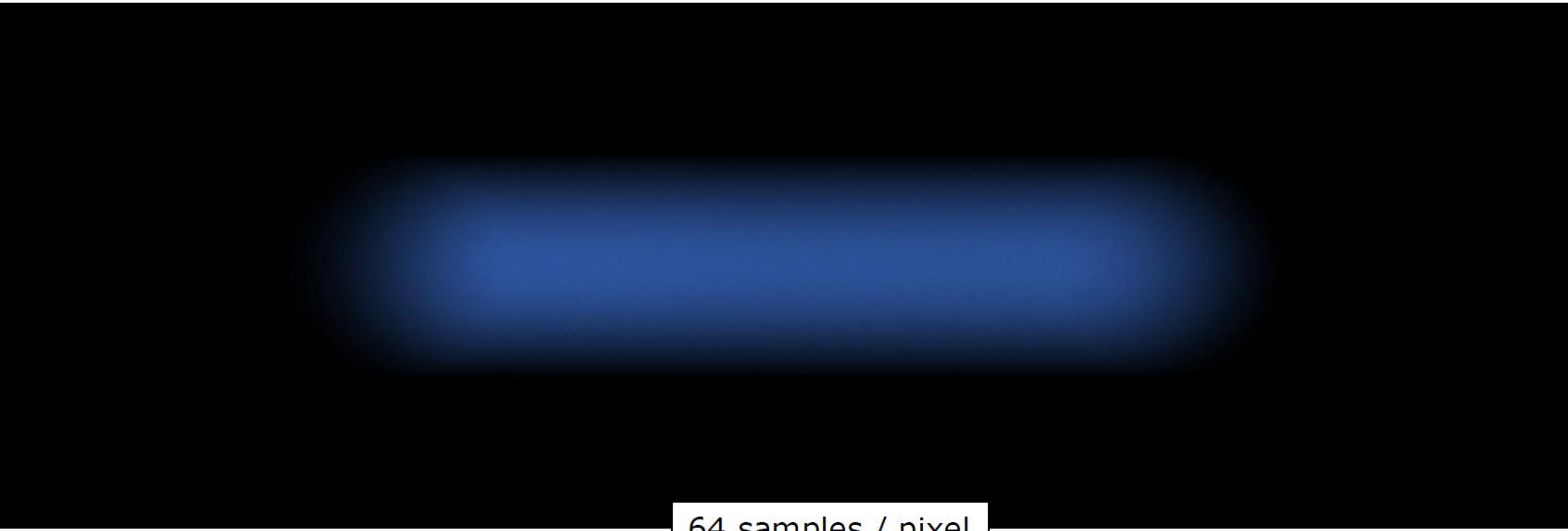


4 samples / pixel

Antialiasing with Stochastic Sampling (3)



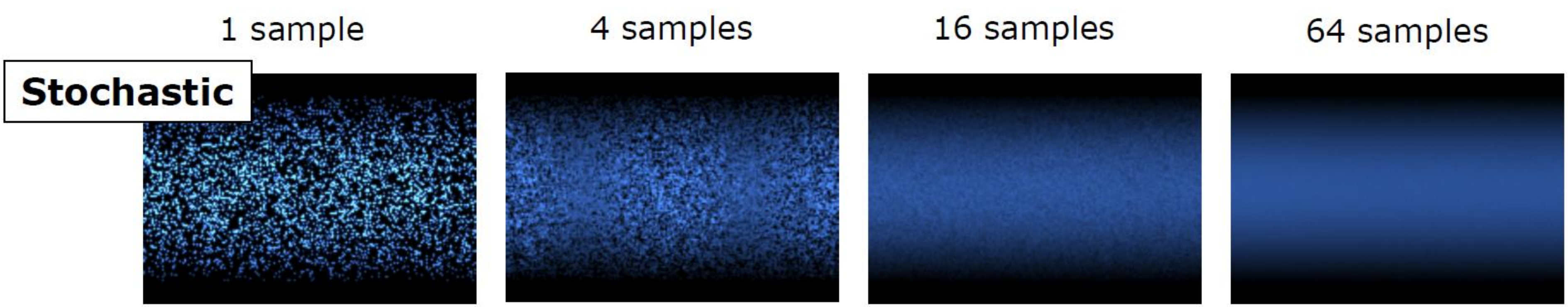
Antialiasing with Stochastic Sampling (4)



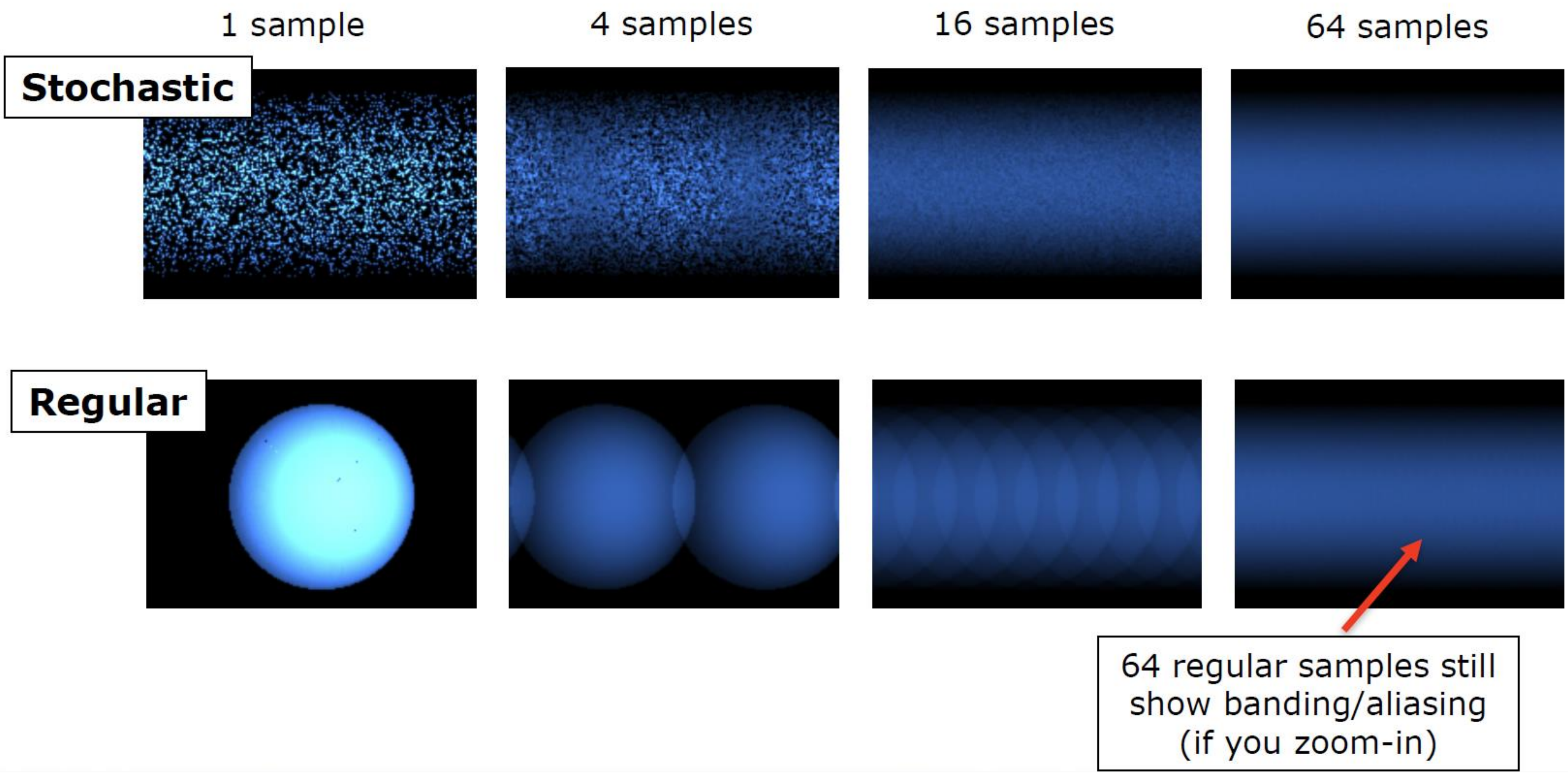
64 samples / pixel

Antialiasing with Stochastic Sampling (5)

4x more samples to get
2x better results (less variance)

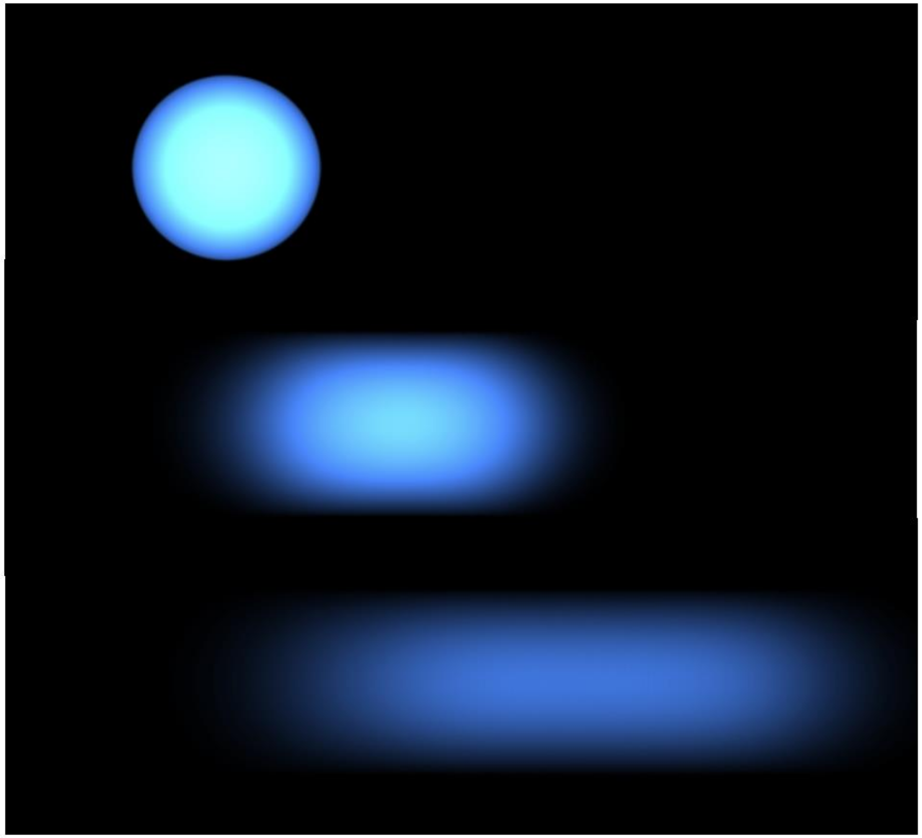


Antialiasing with Stochastic Sampling (6)



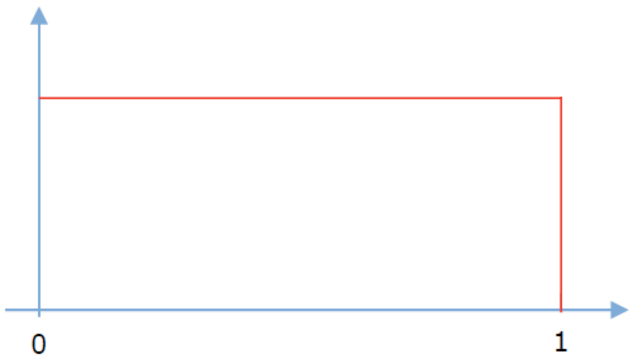
Energy Conservation

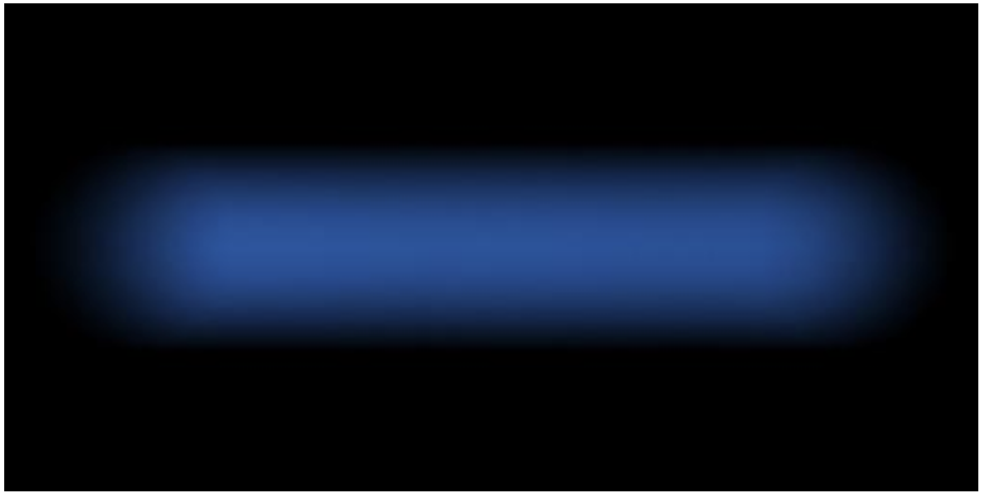
- For a fixed exposure time, speed affects the intensity of the resulting image, as energy is “spread” to larger distances:



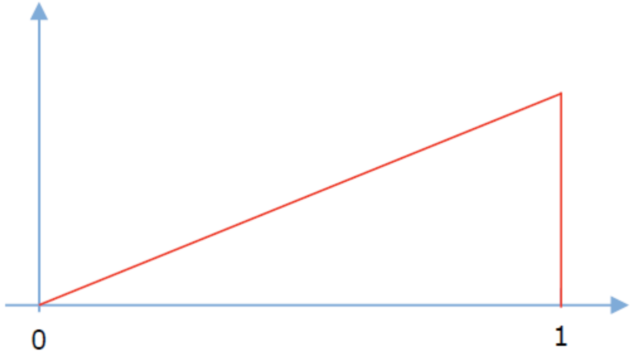
Shutter Profiles (1)

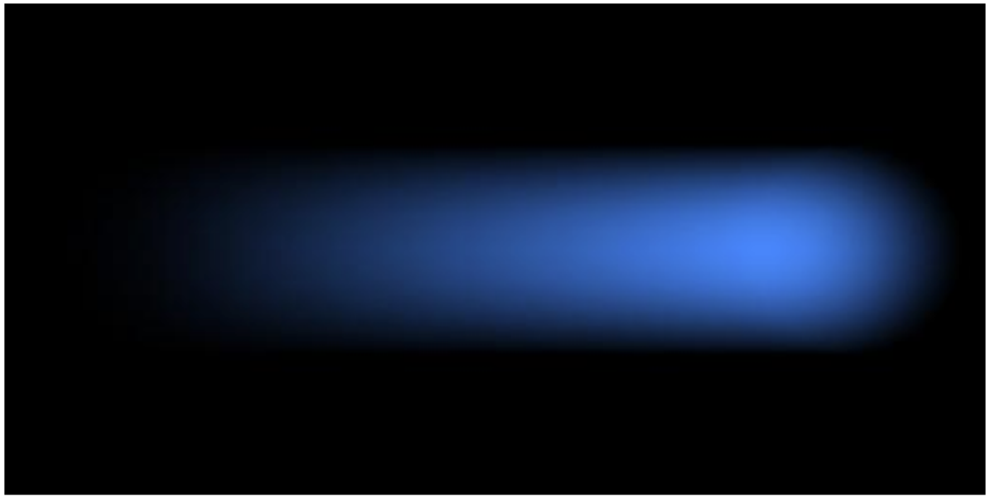
instant open, instant close





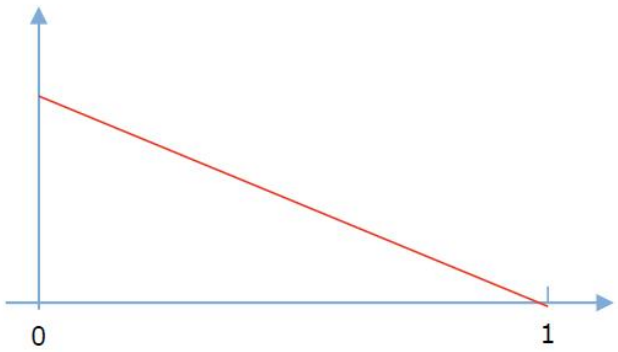
Linear open, instant close

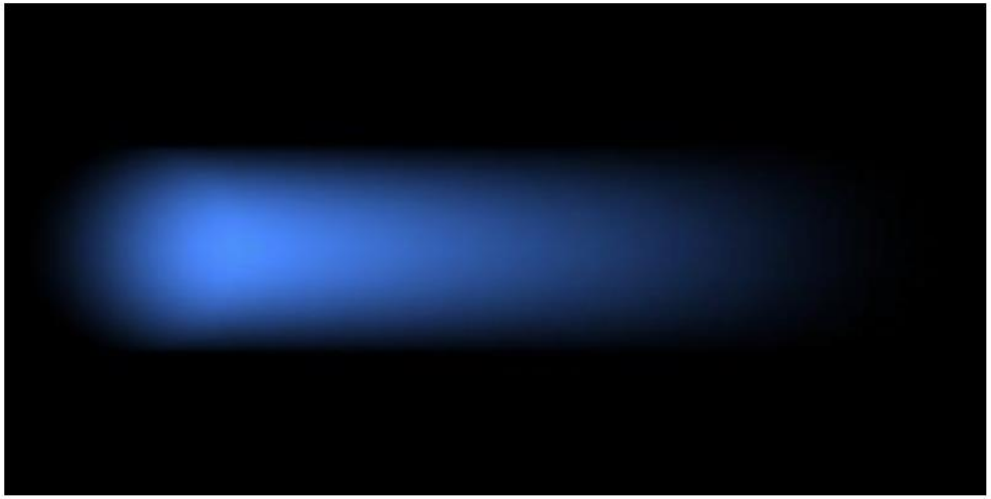




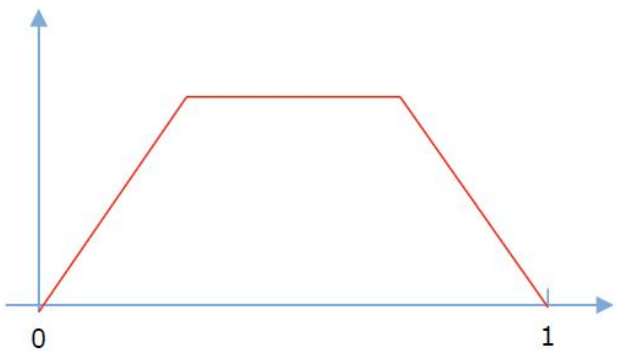
Shutter Profiles (2)

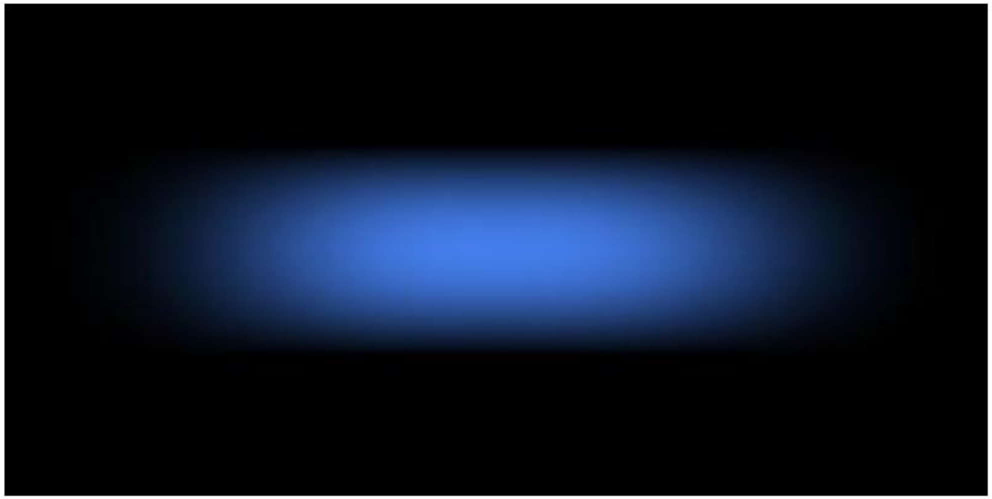
Instant open, linear close



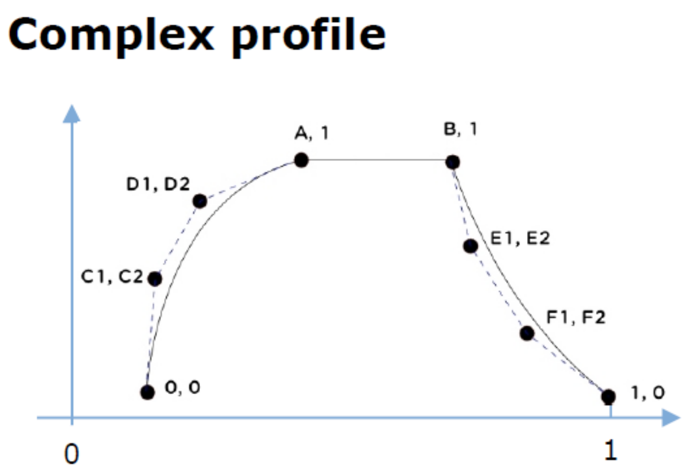


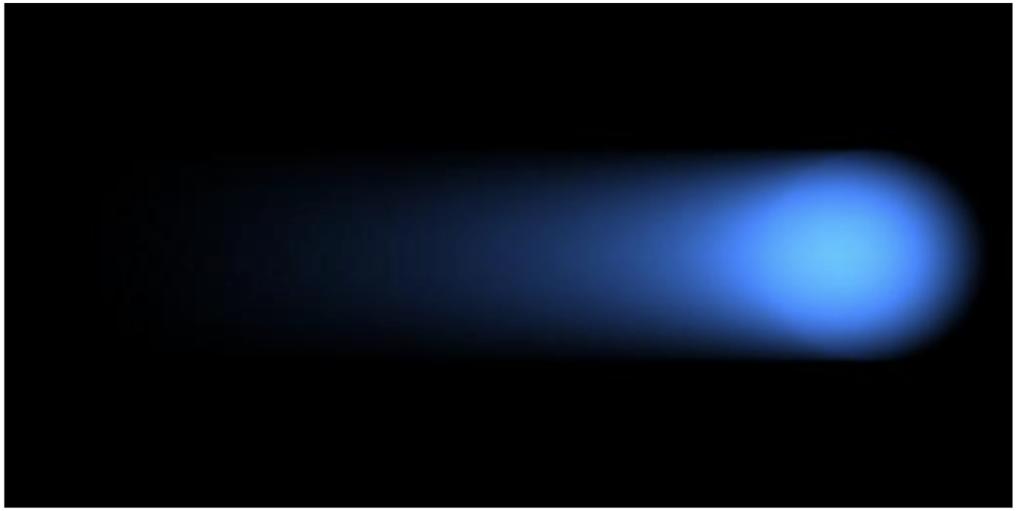
Linear open, linear close





Shutter Profiles (3)

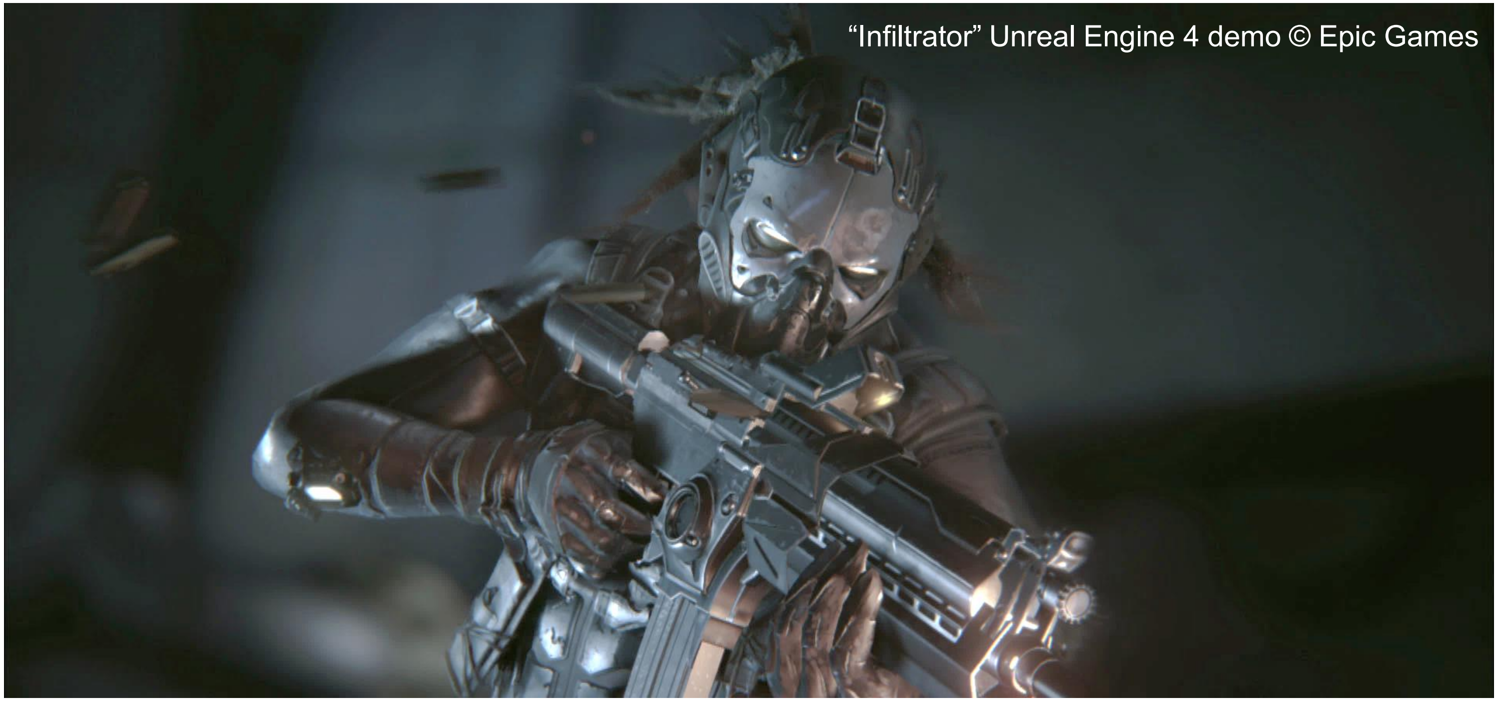




More motion samples towards the end of the shutter interval

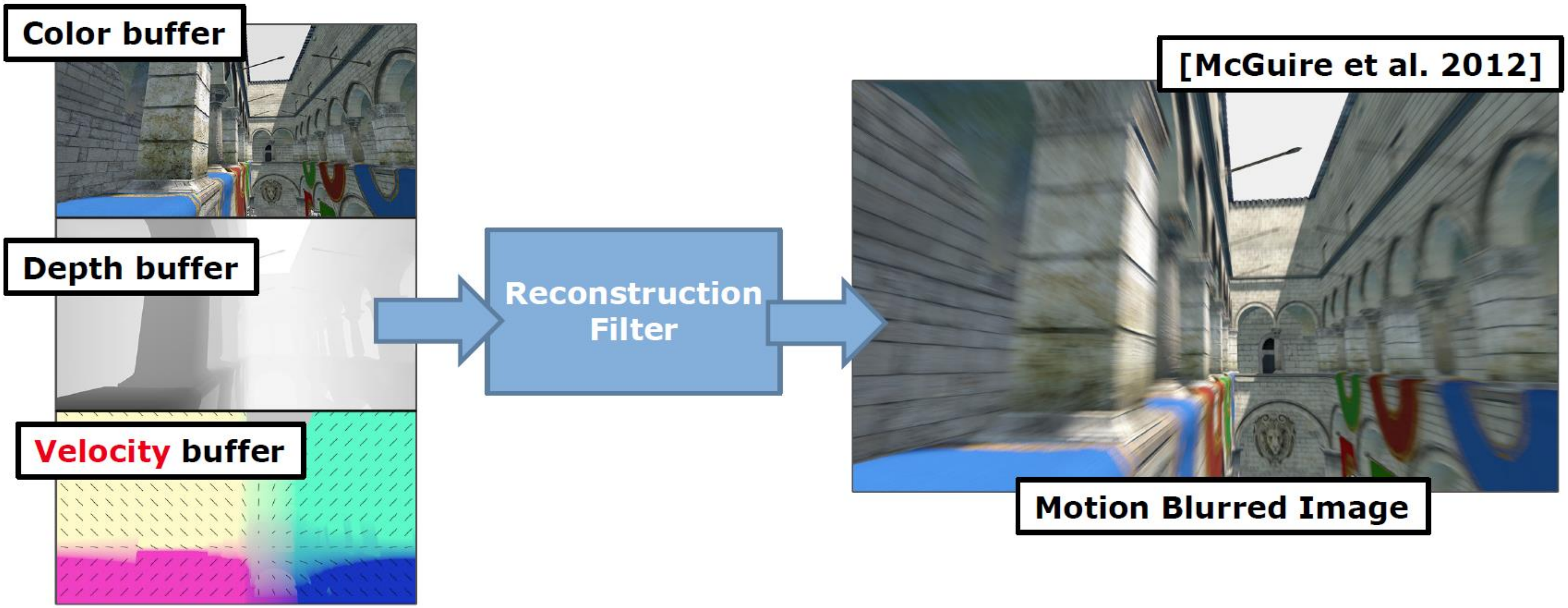
Real-time (RT) Post-filtering

- Re-use samples from previous frames
  - Camera jitter + exponential averaging
  - Motion vectors help recovering fragment position in the past



Motion-blur as Post-process Effect

- Typical solution for **video games** and **real-time** applications



Temporal Pixel Reprojection and Velocity

- Locate the transformed position of the current pixel in the previous frame
  - Retain transformation(s) from the previous frame(s)
  - Transform and interpolate vertices
  - For each pixel obtain transformed positions
  - (optional) store pixel trajectories in velocity buffers

Temporal Pixel Reprojection and Velocity

Depth buffer

Velocity buffer
2 float channels: dx, dy

RT Post-filtering: Re-using Samples

- I found a sample from the previous frame! can I re-use it?
  - Does it come from the right surface?
    - Sample could be from a different object or a mix of objects (e.g. edge → background + foreground)
    - Sample comes from the right object but it has drastically different properties
      - e.g. don’t want to re-use samples across the faces of a cube
  - Did the current fragment even exist in the previous frame?
    - Was partially or completely occluded?
    - POV change?
    - Were we even rendering it? (i.e. popped into existence in the current frame)
  - …

[Salv15]

RT Post-filtering: Artifacts

Pros:
- Very fast run-time
- Easy to integrate in existing applications

Cons:
- Visibility/occlusion is not properly resolved (can result in artifacts, “incorrect” image)

“A boy and his kite” Unreal Engine 4 demo © Epic Games

Contributors

- Georgios Papaioannou
- Theoharis Theoharis
- Pavlos Mavridis

References

[TP*06] T. Theoharis, G. Papaioannou, N. Platis, N. M. Patrikalakis, Graphics & Visualization: Principles and Algorithms, CRC Press

[KC*07] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2007. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (I3D '07). ACM, New York, NY, USA, 39-46. DOI=http://dx.doi.org/10.1145/1230100.1230107

[Kari14] B. Karis, "High Quality Temporal Anti-Aliasing" in "Advances In Real-Time Rendering for Games" Course, SIGGRAPH, 2014

[Salv15] M. Salvi, "Anti-Aliasing: Are We There Yet?" in "Open Problems in Real-time Rendering" Course, SIGGRAPH, 2015