МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

**Нижегородский государственный университет им. Н.И. Лобачевского**

Компьютерная анимация

Учебно-методическое пособие

Рекомендовано методической комиссией факультета иностранных студентов для англоязычных иностранных студентов ННГУ, обучающихся по направлению подготовки 010400 — «Фундаментальная информатика и информационные технологии».

1-е издание

Нижний Новгород  
2015

УДК 004.928

ББК З973.26-018.3

М-29

М-29 КОМПЬЮТЕРНАЯ АНИМАЦИЯ: Автор и составитель Мартынова Е.М. Учебно-методическое пособие. — Нижний Новгород: Нижегородский госуниверситет, 2015.

Рецензент: профессор **В.Е. Турлапов**.

Настоящее пособие содержит англоязычные материалы по основам компьютерной анимации. Предлагается вариант курса “Компьютерная анимация”, методически адаптированный для самоподготовки, включающий в себя краткие конспекты лекций, темы практических занятий и проектов для самостоятельной разработки, а также экзаменационные вопросы.

Учебно-методическое пособие предназначено для англоговорящих иностранных студентов старших курсов, специализирующихся по направлению подготовки 010400 — «Фундаментальная информатика и информационные технологии».

УДК 004.928

ББК З973.26-018.3

MINISTRY OF EDUCATION AND SCIENCE OF RUSSIAN FEDERATION

**Lobachevsky State University of Nizhni Novgorod -**

**National Research University**

Computer Animation

Lecture Notes

These lecture notes are recommended by Methodical Committee of the Department of Foreign Students for English-speaking students of Nizhny Novgorod State University studying at Bachelor's Program 010400 — «Fundamental Informatics and Information Technologies».

First edition

Nizhny Novgorod  
2015

УДК 004.928

ББК З973.26-018.3

M-29

M-29 COMPUTER ANIMATION: Guidance manual. Author and editor - Martynova E.M.— Nizhny Novgorod: State University of Nizhny Novgorod, 2015.

Reviewer: Professor **V.E. Turlapov**.

These guidance manual contains materials in English on fundamentals of Computer Animation. The course cover classical and state of art methods and approaches in Computer Animation. The manual describes content of lectures, topics of practical classes, problems for independent work and examination questions.

The manual is recommended for English-speaking foreign students of the 4th year specializing at Bachelor's Program 010400 — «Fundamental Informatics and Information Technologies».

УДК 004.928

ББК З973.26-018.3

**Section I. Materials (Lecture notes)  
of the course  
“Computer Animation”**

**Lecture 1.** **Introduction to the course.** **Animation principles.**

**Introduction.** Animation is an effective form to engage viewers and makes difficult concepts easier to grasp. Modern animation industry creates films, advertising, games, teaching animation with stunning visual details and quality. This course will investigate state of art methods and algorithms that make these animations possible: keyframing, inverse kinematics, physical simulation of many natural phenomena, motion capture, and data-driven methods.

During the course, students will propose improvements and explore new methods in computer animation by implementation semester-long research projects. Students present their projects in the three steps: extended abstracts, project progress reports, final project presentations and demonstration.

**Animation principles**. Many of the principles of traditional animation were introduced in the 1930's at the Walt Disney studios. These principles were developed to make 2D hand-drawn animation, especially character animation, more realistic and entertaining. These principles can and should be applied to 3D computer animation.

John Lasseter, an American animator, film director, screenwriter, producer and the chief creative officer at Pixar, Walt Disney Animation Studios, and DisneyToon Studios formulated principles in the work *Principles of traditional animation applied to 3D computer animation*, presented at SIGGRAPH 1987.

There are 12 animation principles.

1. Squash and Stretch
2. Timing and Motion
3. Anticipation
4. Staging
5. Follow Through and Overlapping Action
6. Straight Ahead Action and Pose-to-Pose Action
7. Slow In and Out
8. Appeal
9. Arcs
10. Exaggeration
11. Secondary Action
12. Solid Drawing

The lecture proposes useful classification of the animation principles, shown on the figure below.

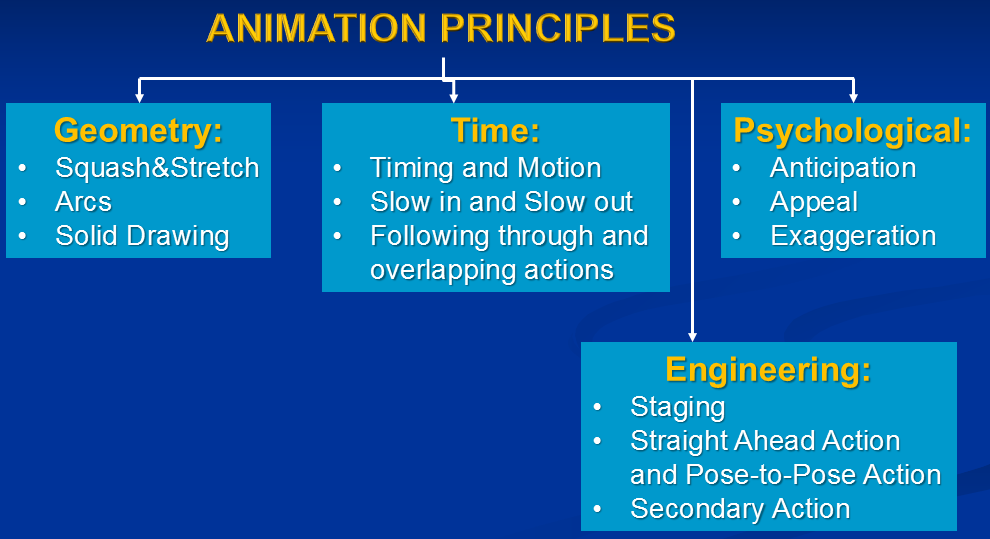


Fig. 1: Classification of animation principles.

**Geometry principles**

* *Squash&Stretch*. Living creatures always deform in shape in some manner. *Squash*: flatten an object or character by pressure or by its own power. *Stretch*: used to increase the sense of speed and emphasize the squash by contrast. An important rule is that the volume of the object should remain constant at rest, squashed, or stretched.

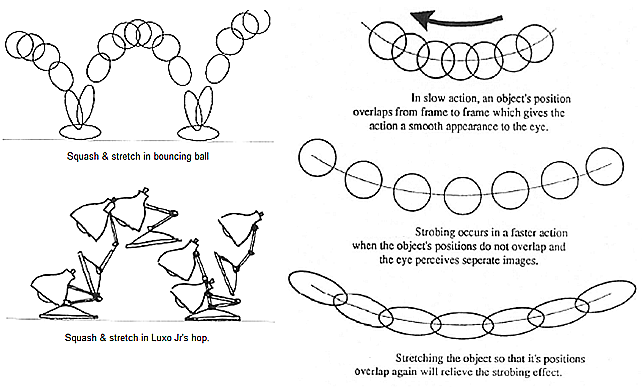


Fig. 2: Examples of Squash & stretch.

These deformations are very important in facial animation: they show the flexibility of the skin and muscle and the relationship between the different facial parts. In very early animation, a character chewing something only moved its mouth and it appeared unrealistic. A later innovation was to have the entire face moving with the mouth motion, thus looking more realistic. This can be exaggerated for effect. A broad smile or frown similarly involves more than the mouth. Squash and Stretch can also be used in the rapid motion of objects: if motion is slow, then the objects overlap between frames and the eye smoothes out the motion. If the motion is too fast, such that there is no object overlap, then the eye sees separate images and the object appears to strobe. A solution is to stretch the object to retain the overlap and smooth motion.

* *Arcs* - the visual path of action for natural movement. Avoid straight lines since most things in nature move in arcs.
* *Solid Drawing*. To improve appearance of animated objects, an artist can add weight, volume and 3D illusion to the subject. This principle was important and innovative during the era of 2D animation.

**Timing principles**.

* *Timing and Motion*. Timing can affect the perception of mass of an object. A heavier object takes a greater force and a longer time to accelerate and decelerate. For example, if a characters picks up a heavy object, e.g., a bowling ball, they should do it much slower than picking up a light object such as a basketball. Similarly, timing affects the perception of object size. A larger object moves more slowly than a smaller object and has greater inertia. These effects are done not by changing the poses, but by varying the spaces or time (number of frames) between poses. Timing can also indicate an emotional state. By varying the number of inbetween frames the meanings of scene can be changed in a wide range.
* *Slow in and slow out*. For example, a bouncing ball moves faster as it approaches or leaves the ground and slower as it approaches leaves its maximum position. The implementation is usually achieved by using splines to control the path of an object. The various spline parameters can be adjusted to give the required effect. In Autodesk 3ds Max this is controlled by the parameters *Ease To* and *Ease From* in the *Key info* window. When these are zero, there is a constant velocity in either direction, i.e., to/from the keyframe. When *Ease To* is set to a higher value, the motion is faster as it leaves the previous keyframe and slows as it approaches the current keyframe. When *Ease From* is set to a higher value the motion is slower leaving the current keyframe and speeds up as it approaches the next keyframe.
* *Following through and overlapping actions*. Here is a quote about overlapping from Walt Disney: "It is not necessary for an animator to take a character to one point, complete that action completely, and then turn to the following action as if he had never given it a thought until after completing the first action. When a character knows what he is going to do he doesn't have to stop before each individual action and think to do it. He has it planned in advance in his mind.”

**Psychological.**

* A properly timed *anticipation* can enable the viewer to better understand a rapid action. Anticipation can be the anatomical preparation for the action, a device to attract the viewer's attention to the proper screen area and to prepare them for the action, staring off-screen at something and then reacting to it before the action moves on-screen. Anticipation can also create the perception of weight or mass, e.g., a heavy person might put their arms on a chair before they rise, whereas a smaller person might just stand up.
* *Appeal* - creating a design or an action that the audience enjoys watching. This is equivalent to charisma in a live actor. A scene or character should not be too simple or too complex. Note: avoid perfect symmetries. The character looks more natural simply because each part of the body varies in some way from the correspondent opposite part.
* *Exaggeration* does not mean just distorting the actions or objects arbitrarily, but the animator must carefully choose which properties to exaggerate: if only one thing is exaggerated then it may stand out too much; if everything is exaggerated, then the entire scene may appear too unrealistic.

**Engineering:**

* *Staging* is the presentation of an idea so that it is clear. An important objective of staging is to lead the viewers eye to where the action will occur. The animator must use different techniques to ensure that the viewer is looking at the correct object at the correct time. Even with modern color 3D graphics, silhouette actions are more clearly delineated and thus to be preferred over frontal action.
* *Straight Ahead Action and Pose-to-Pose Action*. *Straight Ahead Action* is when the animator starts at the first drawing in a scene and then draws all of the subsequent frames until he reaches the end of the scene. This is used for wild, scrambling action. *Pose-to-Pose Action*is when the animator carefully plans the animation, draws a sequence of poses, i.e., the initial, some in-between, and the final poses and then draws all the in-between frames (or another artist or the computer draws the in-between frames). This is used when the scene requires more thought and the poses and timing are important.
* *Secondary Action* is an action that directly results from another action. It can be used to increase the complexity and interest in a scene. It should always be subordinate to and not compete with the primary action in the scene. An example might be the facial expression on a character: the body would be expressing the primary action while the expression adds to it.

Simple example of implementation of animation principles can be found here: <http://www.youtube.com/watch?v=GcryIdriSe4>. The example can be used to get ideas for implementation of animation principles in the independent work.

**References**

1. John Lasseter. Principles of Traditional Animation Applied to 3D Computer Animation (SIGGRAPH 87) //Computer Graphics.-1987.- № 21:4.- pp. 35-44.
2. G. Scott Owen*.* Principles of Traditional Animation Applied to 3D Computer Animation. 1997. - URL: <https://www.siggraph.org/education/materials/HyperGraph/animation/character_animation/principles/prin_trad_anim.htm>.

**Lecture 2.** **Keyframing**

*Keyframing* is an animation technique, in which every frame is controlled: for example, is directly modified or manipulated by the creator, such that no tweening has actually occurred. MPEG-4 facial animation is an example of keyframing. We use the term keyframing for all techniques independently on how often key frames are present in the sequence: each 8th, 4th frame, or each frame.

An example of keyframing is the film of Peter Fouldes - *Hunger* (1974) aka *La faim*, which can be found here: <http://www.youtube.com/watch?v=hY8jpD8zU4Y>.

Transformation between defined key frames can be applied to any animating object parameters: position, shape, velocity, color, lighting settings (light intensity, beam size, light color, and the texture cast by the light). Supposing that an animator wants the beam size of the light to change smoothly from one value to another within a predefined period of time that could be achieved by using key frames. If the beam size value is set at the start of the animation, and for the end of the animation, the software program automatically interpolates the two values, creating a smooth transition.

Key frames can be created by an artist, or defined by specified parameters. In this last case the source of data can be motion capture (will be considered in the next lecture) or any kind of tracking system.

*Inbetween frames* can be calculated by use of:

* Linear interpolation
* Spline interpolation
* Inverse kinematics
* Physical simulation
* Blending

Linear interpolation calculates variables describing keyframes to determine poses for character in between. This is a popular way, but usually it cannot produce data with enough continuity. Spline interpolation in many cases provides better appearance of produced animation.

For example, cubic Hermite interpolator is a spline, where each piece is a third-degree polynomial specified in Hermite form. Interpolating of function at the point *x* inside the interval (*xk*, *xk+1*) can be done with the use of the next function of *t*, which is *t* = (*x* - *xk*)/(*xk+1- xk*):

 (1)

For interpolation inside interval (0,1):

 (2)

Here *pi* – value at the known point *i*, *t* = 0 at *p*0, *t* = 1 at the point *p*1, *mi* is tangent or derivative at the correspondent point. Interpolation on an arbitrary interval can be represented as

, (3)

where *h* are the basis Hermite functions.

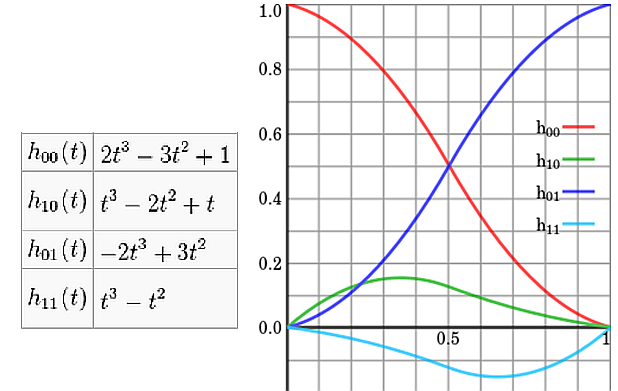
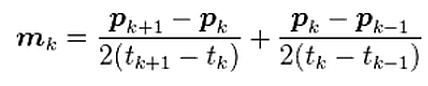
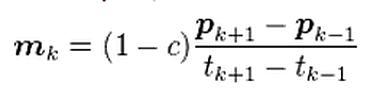


Fig.3: The four Hermite basis functions (the image from Wikipedia, Public domain).

The choice of derivative *mk* is non-unique, and there are several options available. The simplest choice is the three-point difference, not requiring constant interval lengths:

 (4)

A **cardinal spline** uses the tension parameter *c*:

 (5)

To understand application of **inverse kinematics in animation**, first we should describe the way of *articulated figures* representation. In computer animation, an articulated figure is a (often hierarchical) set of rigid segments connected by joints. For example, the human body is represented as a tree of segments, each has linear dimension. Each joint has some Degrees of Freedom (DOF).

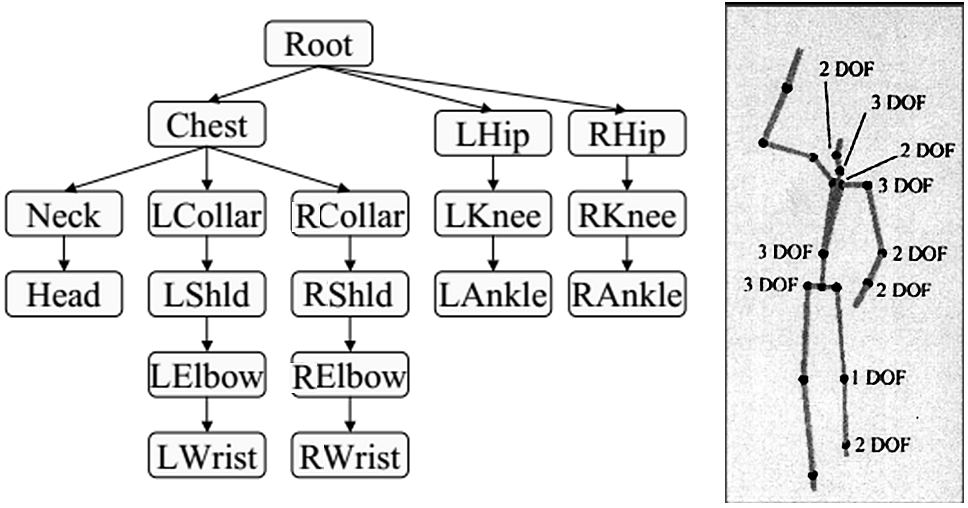


Fig.4: Example of articulated body: human body representation.

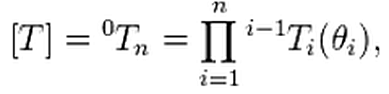
**Forward kinematics** refers to the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters. The kinematics equations for the series chain of a robot are obtained using:

* rigid transformation [Z] to characterize the relative movement allowed at each joint and
* separate rigid transformation [X] to define the dimensions of each link (or segment).

The result is a sequence of rigid transformations alternating joint and link transformations from the base of the chain to its end link, which is equated to the specified position for the end link

 (6)

where [T] is the transformation locating the end-link. The kinematics equations of a serial chain of *n* links, with joint parameters *θi* are given by

 (7)

Where  is the transformation matrix from the frame of link *i* to link *i* – 1.

*Inverse kinematics* (IK) in animation is the process of determining the joint configuration required to place a particular part of an articulated character at a particular location in space. The most popular approach is to incrementally update the joint angles to satisfy the given constraints using Jacobian iteration. In other words, the system gradually pulls the grabbed part to the target location. The resulting pose is dependent on the previous pose, which can easily lead to very unnatural poses. The problem is inherently underdetermined: for example, for given positions of the hands and feet of a character, there are many possible character poses that satisfy the constraints. This is the reason why IK is usually coupled with additional manual (artist correction) or automatic tools of different kind. See the example in [1].

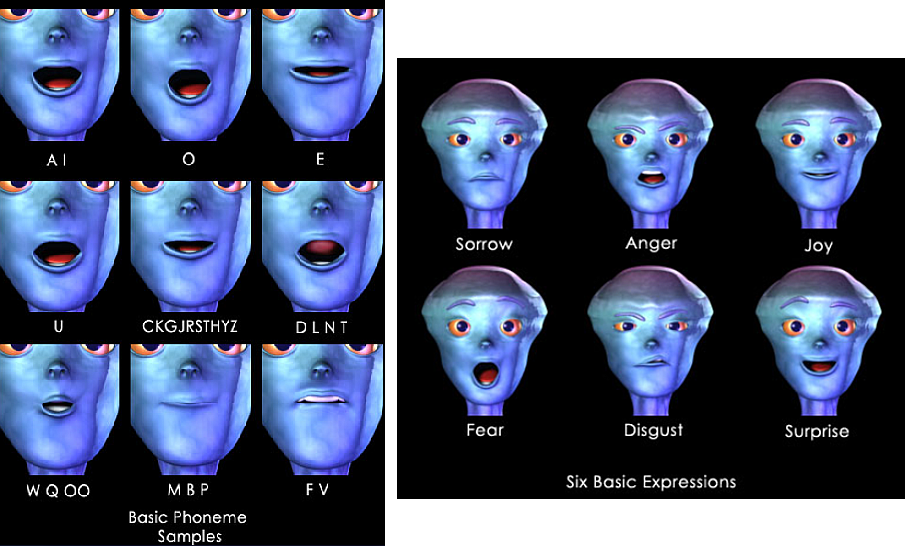


Fig. 5: Phonemes and expressions in Lip Sync. Copyright 1998 Michael B.Comet [4].

*Blend Shapes* are widely used for creation variety of character poses in many animation applications. For example, project [Lip Sync](http://nir3d.com/handouts/Handouts%203D%20Animation%20II%20Applications%20-%20(DIG3354C)/LipSync%20-%20Making%20Characters%20Speak-%20Michael%20B_%20Comet.htm) proposed weighted morphing for creation of phonemes and expressions:

* Different head targets are modeled for phoneme and expression;
* These shapes can be mixed and matched in different percentages yielding a wide variety of poses.

Similar approach is used in MPEG-4 compliant animation (see lectures “Facial Animation”).

**References**

1. Keith Grochow, Steven L. Martin, Aaron Hertzmann, Zoran Popovich. Style-Based Inverse Kinematics // ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH. – 2004. - Volume 23 Issue 3. – pp. 522-531.
2. Rose, Charles F.,III, Peter - Pike Sloan, Cohen, M. F. [Artist-Directed Inverse Kinematic Using Radial Basis Function Interpolation](http://www.ppsloan.org/publications/ikeuro.pdf)*.* // Computer Graphics Forum. – 2001. - № 20, 3. – pp. 239-250.
3. Zhao, Jianmin, and Norman I. Badler. [Inverse kinematics positioning using nonlinear programming for highly articulated figures.](http://ai.stanford.edu/~latombe/cs99k/2000/badler.pdf) // ACM Transactions on Graphics (TOG). – 1994. - №.13, 4. - pp. 313–336.
4. Michael B. Comet. [Lip Sync - Making Characters Speak](http://nir3d.com/handouts/Handouts%203D%20Animation%20II%20Applications%20-%20(DIG3354C)/LipSync%20-%20Making%20Characters%20Speak-%20Michael%20B_%20Comet.htm). - 1998**.** URL:<http://nir3d.com/handouts/Handouts%203D%20Animation%20II%20Applications%20-%20(DIG3354C)/LipSync%20-%20Making%20Characters%20Speak-%20Michael%20B_%20Comet.htm> .

**Lecture 3.** **Motion capture**

Motion capture is the process of sampling the posture and location information of a subject over time. The subject is usually a person, an animal or a machine.

The goal of motion capture is to get the motion data of certain points of interest on the subject, so that either some parameters of the motion (e.g., speed, angle, distance, etc.) can be calculated or the data can be used to control or drive something else.

The application of the data may be motion analysis, biomechanics, sports analysis, biodynamics, etc. In computer animation the data is used to drive a computer generated (CG) character or a scenery to mimic the motion.

*Markerless Motion Capture* provides 3D data by using multiple cameras to simultaneously take multiple images of the subject from different directions, and software to analyze the images. The most difficult task of this approach is recognition of the points of interest. The achievable accuracy of the recognition is insufficient for many applications.

In *Markered Motion Capture* the electro-magnetic, mechanical, gyro, accelerometer and optical fibre based technologies mark the points of interest with *sensors*, while the optical technologies mark the points of interest with *markers*. Markers may be *passive* or *active*. Passive markers do not generate light by themselves (e.g., reflective balls or checker-cross patches), while active markers do.

Motion capture systems are also classified as 'self-contained' or not. The data captured by *self-contained* systems are referenced to the initial posture of the capture subject. The data captured by *non-self-contained* systems are referenced to the parts fixed relative to the ground. Calibration process will need to be done in order to establish the data reference information from time to time. This process involves collecting a relatively large amount of data from the capture space and can be quite tedious.

There are different types of motion capture systems: mechanical, optical, electromagnetic. There exists a set of free motion capture data bases, for example:

* Organic Motion (<http://www.organicmotion.com/motion-capture/> ),
* CMU Graphics Lab Motion Capture Database (<http://mocap.cs.cmu.edu/>),
* [Motion Capture BIP & BVH Library](http://www.motcap.com/).

Motion capture data has proven to be difficult to modify, and editing techniques are reliable only for small changes to a motion. This in particular is a problem for applications that require motion to be synthesized dynamically, such as interactive environments. A set of research was done to overcome this problem.

For example, the work of Kovar and Gleicher [1] presents *motion graph*. Automatic synthesis of directed motion from a corpus of motion capture data provides new sequences of locomotion of different styles. Each pose is represented as a vector of parameters specifying the root position and joint rotations of a skeleton for current frame. The skeleton is only a means to get a final character appearance: in a typical animation, a polygonal mesh is deformed according to the skeleton's pose. To calculate the distance *D(Ai,Bj)* between two motion frames *Ai* and *Bj* the point clouds, formed over two windows of frames of user-defined length *k*, are considered: one bordered at the beginning by *Ai* and the other bordered at the end by *Bj*.

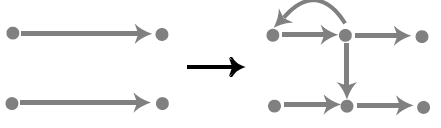


Fig. 6: Motion graph creation [KG02]: inserted node to divide an initial clip into two smaller clips. A transition can join either two different initial clips or different parts of the same initial clip.

The size of the windows are the same as the length of the transitions, so *D(Ai,Bj)* is affected by every pair of frames that form the transition. The distance between *Ai* and *Bj* is calculated by computing a weighted sum of squared distances between corresponding points **pi** and **pi**′ in the two point clouds.

The minimalweighted sum of squared distances given that an arbitrary rigid 2D transformation may be applied to the second point cloud:

 (1)

where the linear transformation  rotates a point **p** about the *y* (vertical) axis by *θ* degrees and then translates it by (*x*0, *z*0). The index is over the number of points in each point cloud. The weights *wi* may be chosen both to assign more importance to certain joints.

If *D(Ai,Bj)* meets the threshold requirements, a transition is creating by blending frames *Ai* to *Ai+k-*1 with *Bj-k+*1 to *Bj*. The first step is to apply the appropriate aligning to 2D transformation to motion *B*.

Then on frame *p* of the transition (0 <= *p* < *k*) we linearly interpolate the root positions and perform spherical linear interpolation on joint rotations:

 (2)

, (3)

where *Rp* is the root position on the *pth* transition frame and *qip* is the rotation of the *ith* joint on the *pth* transition frame, α(*p*) are the blend weights.

To maintain continuity the blend weights α(*p*) are choosing according to the conditions that α(*p*) = 1 for p<= - 1, α(*p*) = 0 for *p*>=*k*, and that α(*p*) has *C*1 continuity everywhere. This requires

 (4)

Other transition schemes may be used in place of this one. The figure 6 demonstrates the fragment of motion capture graph with transition between appropriate motion frames.

**References**

1. KOVAR, L., GLEICHER, M., PIGHIN, F. Motion Graphs. (Proc. SIGGRAPH 2002) // ACM Transactions on Graphics. – 2002. - № 21, 3 (July). – pp. 473–482.
2. KOVAR, L., GLEICHER, M. Automated Extraction and Parameterization of Motions in Large Data Sets // ACM Transactions on Graphics. – 2004. - № 23, 3 (Aug.) – pp. 559-568.
3. Wiley, D. J., and Hahn, J. K. Interpolation synthesis of articulated figure motion // IEEE Computer Graphics and Applications. – 1997. - № 17, 6. – pp. 39–45.
4. Zordan, V. B., Majkowska, A., Chiu, B., Fast, M., *Dynamic Response for Motion Capture Animation,* ACM Transactions on Graphics (ACM SIGGRAPH 2005), № 24, 3, 697-701.
5. Müller M., Röder, T., Clausen, M. Efficient Content-Based Retrieval of Motion Capture Data (Proceedings of ACM SIGGRAPH 2005) //ACM Transactions on Graphics. - 2005. - № 24(3). - pp. 677-685.

**Lecture 4.** **Particle Systems.**

Particle systems model an object as a cloud of primitive particles that define its volume. The use of Particle systems is a way of modeling fuzzy objects, such as fire, clouds, smoke, water, etc. Stochastic processes are used to generate and control the many particles within a particle system.

Though stochastic processes are used to create and change an object's shape and appearance, particle systems can be used in a deterministic way to create certain objects, e.g., the human head in the video ["Particle Dreams" by Karl Sims](http://www.siggraph.org/education/materials/HyperGraph/video/video_references.htm).

A particle system is a collection of many minute particles that model some object. For each frame of an animation sequence the following steps are performed:

* New particles are generated;
* Each new particle is assigned its own set of attributes;
* Any particles that have existed for a predetermined time are destroyed;
* The remaining particles are transformed and moved according to their dynamic attributes;
* An image of the remaining particles is rendered.

Particles are *generated* by means of different controlled processes. For example, in one method the designer controls the mean number of particles generated per frame and the variance. So the number of particles generated at frame *F* is

*NpartsF* = *MeanPartsF* + *Rand*() × *VariancePartsF* (1)

where *Rand*() is a uniformly distributed random number between -1.0 and + 1.0, *MeanPartsF* - the mean number of particles, and *VariancePartsF* - its variance.

A second method generates a certain number of particles per screen area. So *MeanParts* and *VarianceParts* refer to a number per unit screen area:

*NpartsF* = (*MeanPartsSAF* + *Rand*() × *VariancePartsSAF*) × *ScreenArea* (2)

*SAF* means per Screen Area for frame *F*. This method is good for controlling the level of detail required.

The number of particles generated as time changes and is controlled by a simple linear function:

*MeanPartsF* = *InitialMeanParts* + *DeltaMeanParts* ×(*F-F*0) (3)

The designer could do this by some function other than linear if needed or desired.

Each new particle has the following *attributes*:

* initial position
* initial velocity (both speed and direction)
* initial size
* initial color
* initial transparency
* shape
* lifetime.

A particle system has several parameters that control the **initial position** of the particles: *X*, *Y*, *Z* (the particle system origin), two angles of rotation that give its orientation. The **generation shape** describes the initial direction of new particles:

* for a sphere the particles would move away from the origin in all directions.
* for a planar shape, e.g. a circle in the x-y plane, the particles would move up and away from the plane.

The initial **speed** of a particle can be given by:

*InitialSpeed* = *MeanSpeed* + *Rand*() × *VarSpeed* (4)

The initial **color** can be:

*InitialColor* = *MeanColor* (*R,G,B*) + *Rand*() × *VarColor(R,G,B*) (5)

The initial **opacity** can be:

*InitialOpacity* = *MeanOpacity* (*R,G,B*) + *Rand*() × *VarOpacity(R,G,B*) (6)

The initial **size** can be:

*InitialSize* = *MeanSize* + *Rand*() × *VarSize* (7)

There is also a parameter that specifies the **shape** of each particle, e.g., spherical, rectangular, or streaked spherical (for motion blur).

Individual particles within a particle system **move in three-dimensional** space and also change over time in color, transparency, and size. The rates of change can be global for all particles in a particle system, or this parameter can be made stochastic.

When it is generated, a particle is given a **lifetime** measured in frames. This lifetime is decremented after each frame. A particle is killed when its lifetime reaches zero. Other mechanisms, if enabled, arrange for particles to be killed because they can contribute nothing to the image. If the intensity of a particle, calculated from its color and transparency, drops below a specified threshold, the particle is killed.

The general **particle rendering** problem is as complicated as the rendering of objects composed of the more common graphical primitives, such as polygons and curved surfaces.

* Particles can obscure other particles that are behind them in screen depth.
* They can be transparent and can cast shadows on other particles.
* Furthermore, particles can coexist in a scene with objects modeled by surface-based primitives, and these objects can intersect with the particles.

Objects modeled using other techniques are composited together with particle system objects in a post processing compositing stage.

A **hierarchy** can be used to exert global control on a complicated fuzzy object that is composed of many particle systems. The parent particle system's mean color and its variance are used to select the mean color and variance of the offspring particle systems using the same equations. The number of new particle systems generated at a frame is based on the parent's particle generation rate.

An example: **Wall of Fire** from *The Genesis Demo sequence* from the movie *Star Trek II: The Wrath of Khan,* which was generated by the Computer Graphics project of Lucasfilm Ltd.

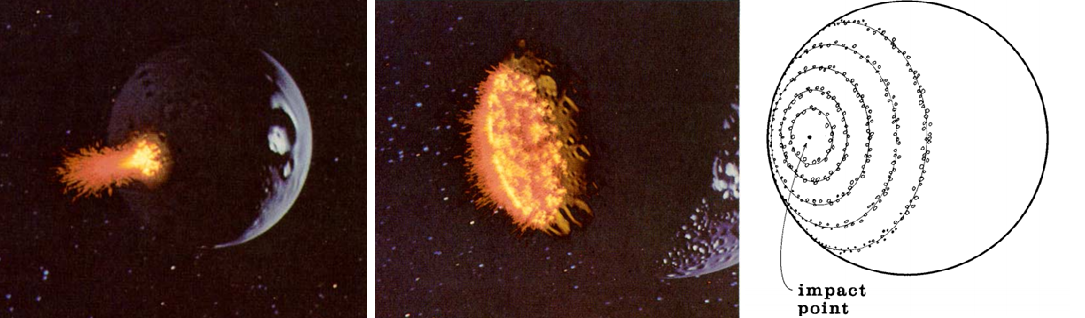


Fig.7: Wall of Fire on the surface of planet. The algorithm of particle creation in the two-level system.

The top-level system generated particles which were themselves particle systems. Figure 7 illustrates the positions of these second-level particle systems and how they formed expanding concentric rings on the surface of the planet. The number of new particle systems generated in each ring was based on the circumference of the ring and a density parameter. New particle systems were spaced randomly around the ring.

The next example is a particle system for simulation of firework and grass. In Firework the control parameters of the particle systems vary more widely, and streaking is more predominate. Figure 8a contains overlapping, multicolored explosions formed with different generation shapes and ejection angle.

To model grass instead of drawing particles as little streaks, the parabolic trajectory of each particle over its entire lifetime is drawn. Thus, the time-domain motion of the particle is used to make a static shape. Grasslike green and dark green colors are assigned to the particles, which are shaded based on the scene's light sources.

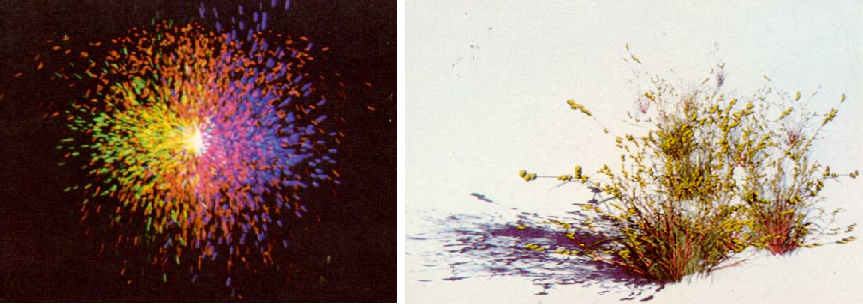


Fig.8: Use of particle systems for (a) – Firework, (b) – grass (*white.sand* by Alvy Ray Smith of Lucasfilm).

**References**

1. Reeves, W.T. [Particle Systems – A Technique for Modeling a Class of Fuzzy Objects](http://zach.in.tu-clausthal.de/teaching/vr_literatur/Reeves%20-%20Particle%20Systems.pdf) // ACM SIGGRAPH Computer Graphics. - 1983. - № 17 (3). – pp. 359-375.
2. Karl Sims. [Particle Animation and Rendering Using Data Parallel Computation.](http://www.karlsims.com/papers/ParticlesSiggraph90.pdf) (SIGGRAPH '90 Proceedings)// ACM SIGGRAPH Computer Graphics. – 1990. - № 24 (3). – pp. 405–413.
3. Andrew Witkin. [Particle System Dynamics.](http://www.pixar.com/companyinfo/research/pbm2001/pdf/notesc.pdf) // ONLINE SIGGRAPH 2001 COURSE NOTES.- 2001. – 13 p. URL: <http://www.pixar.com/companyinfo/research/pbm2001/pdf/notesc.pdf>.

**Lecture 5.** **Rigid Bodies.**

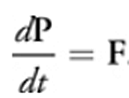
In physics, a *rigid body* is an idealization of a solid body in which deformation is neglected. In other words, the *distance* between any two given points of a rigid body *remains constant in time* regardless of external forces exerted on it.

In classical mechanics a rigid body is usually considered as a *continuous mass distribution*, while in simulation a rigid body is usually thought of as a *collection of point masses*).

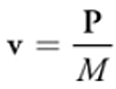
In applications like computer games, it is necessary to calculate the effect of an external **unpredictable** impact and the problem must be resolved **real time.** Therefore, the process of simulation must use **physical models**.

The motion of a rigid body contains two components: *translation* and *rotation.*

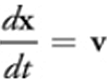
When a force F acts on a rigid body, this force can cause a variation of its *linear momentum P*. More precisely, the time derivative of P is equal to the force F:

 (1)

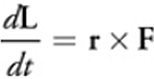
The *velocity* v of the center of mass from P and the rigid body's mass *M*, can be described using the following formula:

 (2)

The velocity is the time derivative of the position x of the *center of mass*:

 (3)

A force F that acts at some point of a rigid body that is different from the center of mass can also cause a variation of the rigid body's *angular momentum* **L**. This variation depends on the relative position r of the acting point to the center of mass. The time derivative of **L** is the torque that is the cross-product of **r** and **F**:

 (4)

The *angular velocity* w, whose norm is the spinning speed, is derived from the angular momentum **L**

  (5)

where the 3x3 matrix I(t) is the *inertia tensor* at time t and I(t)-1 is its inverse. The inertia tensor is a physical value of a rigid body, which indicates its resistance against rotational motion.

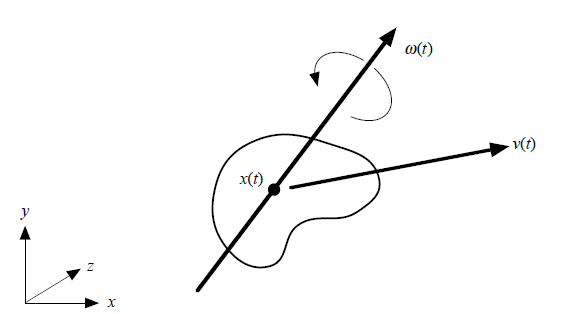


Fig. 9: The center of mass of a rigid body, liner velocity and angular velocity.

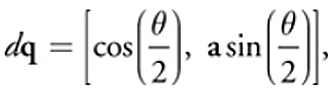
Because the inertia tensor changes as the orientation of a rigid body changes, it must be reevaluated at every time step. To calculate **I**(*t*)-1, the rotation matrix **R**(*t*) is used. The matrix describes the orientation of a rigid body at time *t* and may be calculated from a quaternion.

**I**(*t*)-1 is obtained from the rotation matrix at time *t* as follows:

 (6)

A quaternion **q** = [*s*, *vx* , *vy* , *vz* ] represents a rotation of *s* radians about an axis

(*vx* , *vy* , *vz* ). It can be also represented as **q** = [*s*, **v**], where **v** = (*vx* , *vy* , *vz* ). The variation of quaternion *d***q** with angular velocity **w** is calculated as the following:

 (7)

where **a** = w/|w| is the rotation axis and *θ* = |w *d*t| is the rotation angle.

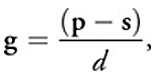
The quaternion at time *t* + *dt* is updated as follows:

 (8)

Rigid bodies’ interaction is a computationally hard task. Different methods of used optimization apply some kind of preprocessing to obtain possibility of fast estimation of mutual objects position with minimal amount of calculation.

As an example we consider the method, oriented to the GPU usage. It applies representation of rigid bodies as a set of particles. To generate the particles inside a rigid body, the space around the rigid body is discretized by defining a 3D grid that encloses it, and one particle is assigned for each voxel inside the rigid body. Now collision detection between rigid bodies with complicated shapes means collisions between particles. When the distance between two particles is smaller than their diameter, collision occurs.

The applied three-dimensional uniform grid covers the whole computational domain, and the domain is divided into voxels. The voxel index **g** = (*gx* , *gy* , *gz* ) of a particle at **p** = (*px* , *py* , *pz* ) is calculated as follows:

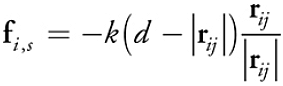
 (9)

where **s** = (*sx* , *sy* , *sz* ) is the grid corner with smallest coordinates and *d* is the side length of a voxel.

With the grid, particles that collide with particle *i* are restricted to collide only with adjacent particles. Consequently, the efficiency is improved from *O*(*n*2) to *O*(*n*). Computation becomes most efficient when the side length of a voxel is set to the diameter of particles, because the number of voxels to search is the smallest: 32 in two dimensions and 33 in three dimensions.

The interparticle forces between colliding pairs are calculated by applying the *discrete element method* (DEM), which is a method for simulating granular materials (see [1]).

A repulsive force **f** *i*,*s* , modeled by a linear spring, and a damping force **f** *i*,*d* , modeled by a dashpot—which dissipates energy between particles—are calculated for a particle *i* colliding with a particle *j* as the following:

 (10)

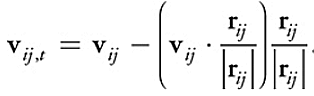
 (11)

where *k*, *h*, *d*, **r** *ij* , and **v** *ij* are: spring coefficient, damping coefficient, particle diameter, and relative position and velocity of particle *j* with respect to particle *i*, respectively.

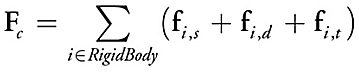
A shear force **f** *i,t* is also modeled as a force proportional to the relative tangential velocity **v** *ij*,*t* :

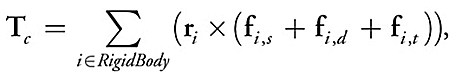
 (12)

where the relative tangential velocity is calculated as

 (13)

The force and torque applied to a rigid body are the sums of the forces exerted on all particles of the rigid body:

 (14)

 (15)

where **r** *i* is the current relative position of particle *i* to the center of mass.

Having this basic algorithm of rigid body simulation, an iteration consists of the following five stages:

* Computation of particle values
* Grid generation
* Collision detection and reaction
* Computation of momenta
* Computation of position and quaternion.

The collision detection algorithm begins with a preprocessing step, in which a bounding box for each rigid body is computed (a box with sides parallel to the coordinate axes). Given *n* such bounding boxes, it is possible to quickly determine all pairs of bounding boxes that overlap. Pairs of rigid bodies whose bounding boxes do overlap require further consideration. If inter-penetrating was found, this indicates a collision at some earlier time, the simulator will back up and attempt to compute *d***x***/dt* at some earlier time.

To reduce the number of pairwise collision/contact determinations necessary, a bounding box hierarchy can be applied on the bodies in the simulation environment. If two bounding boxes are found not to overlap, no further comparisons involving the contents of the boxes are needed. Given a collection of *n* rectangular bounding boxes, aligned with the coordinate axes, it is possible to efficiently determine all pairs of boxes that overlap. A naive pairwise comparison of all pairs requires *O(n*2)work and is too inefficient, unless the number of bodies is small. Computational geometry algorithms exist that can solve this problem in time *O(nlogn+k),* where *k* is the number of pairwise overlaps.

A set of data structures and correspondent algorithms can be applied to the problem of collision detection:

* k-d Tree,
* Bounding Volume Hierarchy (BVH),
* Binary Space-Partitioning Tree (BSP Tree),
* V-Clip,
* Lin-Canny.

“One of the most effective method for determining intersection between polyhedral” is an iterative GJK algorithm (due to Gilbert, Johnson, and Keerthi). The algorithm and many related questions are described in the book [Ericson Christer. Real-Time Collision Detection PDF](http://www.twirpx.com/file/188040/) *Morgan Kaufmann*, 2004. 632 pages.

**References**

1. Takahiro Harada. Real-Time Rigid Body Simulation on GPUs. // GPU Gems-3. – 2007. - Chapter 29. URL: <http://http.developer.nvidia.com/GPUGems3/gpugems3_ch29.html>.
2. David Baraff. [Physically Based Modelling. Rigid Body Simulation.](http://www.pixar.com/companyinfo/research/pbm2001/pdf/notesg.pdf)  // ONLINE SIGGRAPH 2001 COURSE NOTES.- 2001. – 69 p. - URL: <http://www.pixar.com/companyinfo/research/pbm2001/pdf/notesg.pdf>
3. M. Nießner, C. Siegl, H. Schäfer and C. Loop. [Real-time Collision Detection for Dynamic Hardware](http://research.microsoft.com/en-us/um/people/cloop/collision.pdf) Tessellated Objects // Eurographics 2013 - Short Papers Proceedings, Girona, Spain - 2013. – pp. 33-36.

**Lecture 6. Legged Locomotion.**

A problem with modeling of walking or running figure is in big number of degrees of freedom, when the coordination of legs, body and feet are functionally related in a complex fashion. The motion of the body and the timing and placement of legs are both kinematically and dynamically coupled.

A **gait** refers to a particular sequence of lifting and placing the feet during legged locomotion (gallop, trop, walk, run…). Each repetition of the sequence is called the **gait cycle**. The time taken to complete a single gait cycle is the period P of the cycle. The inverse of the period is the **gait frequency** (1/period).

The relative phase of leg *i*, *Ri* , describes the fraction of the gait cycle period which transpires before leg *i* is lifted. The relative phases of the legs may be used to classify the well known gaits of quadrupedal animals. During each gait cycle period any given leg will spend a percentage of that time on the ground-this fraction is called the **duty factor** of leg *i.*

The **step phase** is a value that ranges from 0 to 1 during an individual leg’s step cycle. The foot contacts the ground and comes to rest when the phase equals 1 minus the duty factor. Each leg’s step cycle is phase shifted relative to the main gait cycle.

This phase shift is called the **step trigger.** The **trigger** is the phasewithin the main gait cycle where a particularleg*begins its step cycle***.**

A simple description of the timing of a particular gait requires the following information, which can be represented on the scheme (see the left figure below):

* Number of legs,
* Gait period,
* Duty factor & step trigger for each leg.

Edward James Muybridge (1830 – 1904) was an English photographer important for his pioneering work in photographic studies of motion. Muybridge showed that almost all quadrupeds (animals with 4 legs) use one or more of the following gaits:

* Walk,
* Amble,
* Trot,
* Rack/Pace,
* Gallop (rotary & transverse),
* Canter.

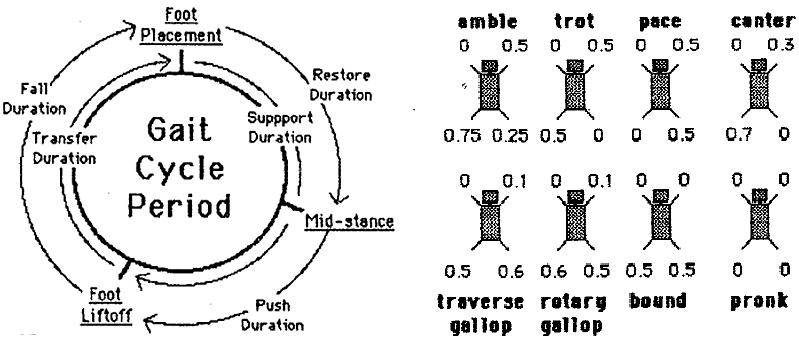
****

Fig.10: Locomotion terminology and schematic gait description.

In an example implementation of legged locomotion system (Steve Rotenberg, [“Legged Locomotion”](http://digitool.library.colostate.edu/exlibris/dtl/d3_1/apache_media/L2V4bGlicmlzL2R0bC9kM18xL2FwYWNoZV9tZWRpYS8xNjkwNTk=.pdf)) a set of classes to control the simulation process and the main algorithm are described:

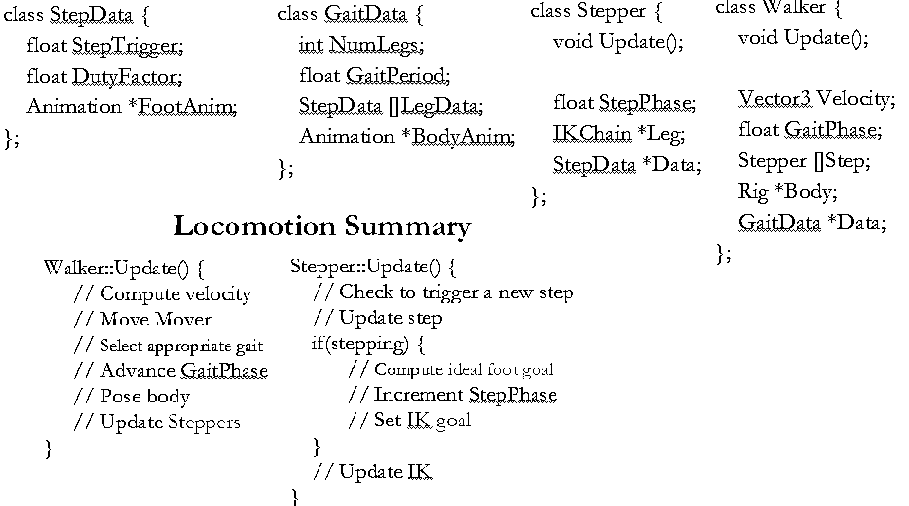
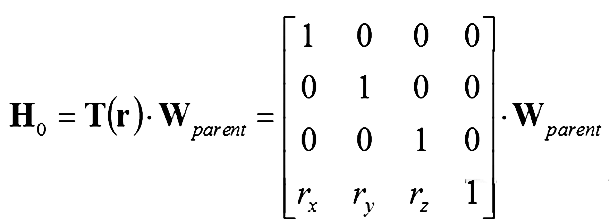


Fig.11: Locomotion data structure and simulation procedure description.

Let’s consider an example of Analytical Inverse Kinematics application to the 3-DOF leg.

A matrix to describe hip position is: a world matrix, representing the hip position in an unrotated state, multiplied to the translation matrix for the hip offset:

 (1)

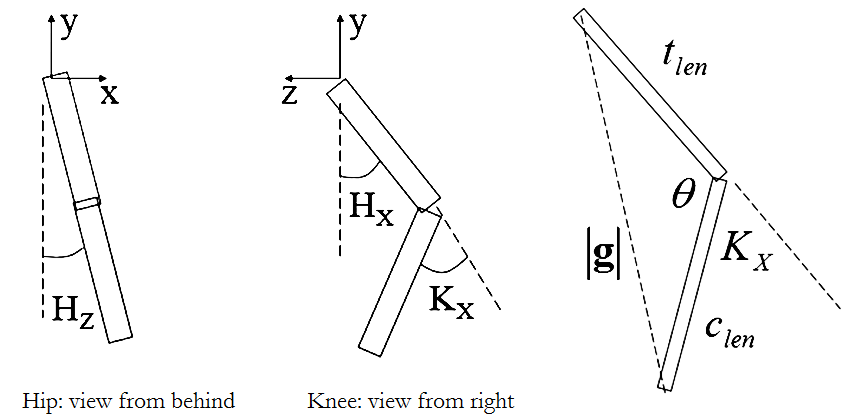
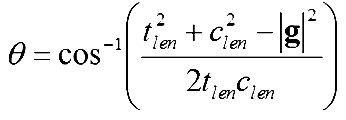


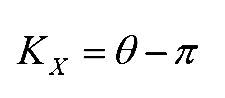
Fig.12: Example of Analytical Inverse Kinematics: 3-DOF leg from z and x directions; the triangle, formed by two limbs.

To transform the IK goal position relative to the unrotated hip space, it is possible to solve the problem in this space:

 (2)

The length of the thigh & calf are assumed to be constant. They make up two sides of a triangle. The third side of the triangle is made by the distance from the hip to the goal. As the hip pivot is located at [0 0 0] in hip space, the distance is the magnitude of **g.** From the*law of cosines* it is possible to find the knee angle:

 (3)

 (4)

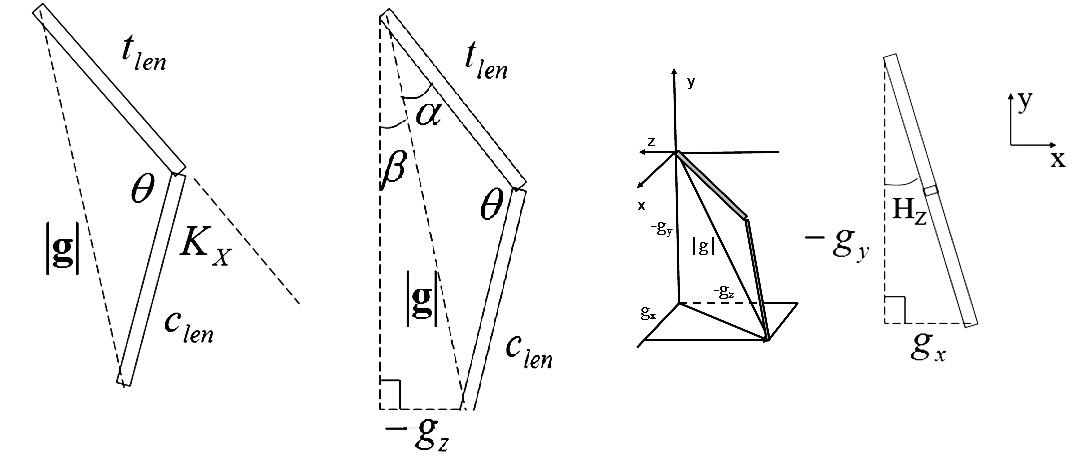
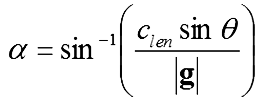
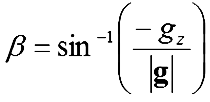


Fig.13: The triangle, based on hip and knee, and calculated values.

The next step is a calculation of the hip X rotation. Further triangle analysis allows, using the *law of sines*, to find the upper angle α in the triangle, and then add that to the angle β to the goal.

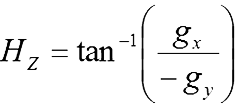
The problem is considered in the plane normal to the knee rotation axis.

 (5)

 (6)

. (7)

To find the hip z angle the goal position (in hip space) is considered in the XY plane:

 (8)

In common case the process is more complicated, as some of the equations may result in divide by zero’s, square roots of negative numbers, or inverse trig functions with parameters outside of the legal range. These cases indicate situations where there is no solution and may imply problems such as:

* Goal out of reach (further than *tlen*+*clen*)
* Goal too close (closer than | *tlen* - *clen* |)

These cases should be checked and appropriate alternative solutions need to be designed to handle them.

**References**

1. M. Girard, A. Maciejewski. [Computational Modeling for the Computer Animation of Legged Figures,](http://digitool.library.colostate.edu/exlibris/dtl/d3_1/apache_media/L2V4bGlicmlzL2R0bC9kM18xL2FwYWNoZV9tZWRpYS8xNjkwNTk=.pdf) // SIGGRAPH '85 Proceedings of the 12th annual conference on Computer graphics and interactive techniques. - 1995. - pp. 263-270.
2. Steve Rotenberg. “Legged Locomotion” //The University of California, San Diego. CSE169: Computer Animation. - 2005. - URL: <https://www.google.ru/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CBwQFjAA&url=http%3A%2F%2Fgraphics.ucsd.edu%2Fcourses%2Fcse169_w05%2FCSE169_14.ppt&ei=c-NKVeyRDYfWygOno4AQ&usg=AFQjCNG3x61F7_gzLh0z8XHZt0dhl5MQCw&sig2=z4sj2GoFUfneUJnvCfWLLA&bvm=bv.92291466,bs.1,d.bGQ>
3. [K.Wampler, and Z. Popović](http://grail.cs.washington.edu/projects/animal-morphology/s2009/Optimal_Gait_and_Form_for_Animal_Locomotion.pdf). [Optimal gait and form for animal locomotion.](http://grail.cs.washington.edu/projects/animal-morphology/s2009/Optimal_Gait_and_Form_for_Animal_Locomotion.pdf)  // ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2009. - № 28, 3. – p. 8.

**Lecture 7-8. Face Animation.**

To animate human face, it is necessary to know the anatomy of human head. In the process of simulation a head is represented as a set of objects: skull, facial muscles, skin, eyes and eyelashes, teeth, mouth cavity, tongue, hair. Human skull contains about 200 bones. Human face has 57 muscles of three types: sphincters, liner muscles and sheet muscles.

The experience of human head animation was accumulated in the special section of standard MPEG-4: ISO/IEC 14496-2 (“Visual”). It introduced parameters, which allowed: to describe face, to control animation and to transfer animation sequences from some face to another one.

The parameters are:

FDP – Face Definition Parameters,

FP – Feature Points,

FAP – Face animation parameters,

FAPU – Face Animation Parameter Units.

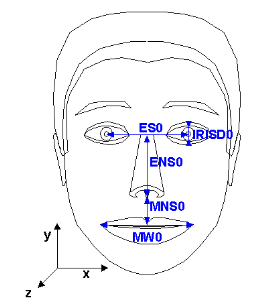
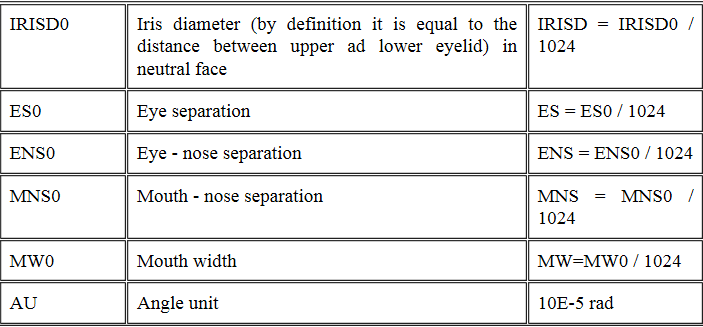
 

Fig.14: Face Animation Parameter Units.[1].

**Face Animation Parameter Units** is a set of units for facialanimation parameters. The use of FAPU provides possibility to apply animation sequences, obtained for some face, to any other faces, having the same set of feature points.

MPEG-4 specifies 84 **feature points** on the neutral face (black ones are affected by FAPs (facial animation parameters). In order to control animation of all objects in the head model the feature points were specifies also for eyes, teeth, tongue.

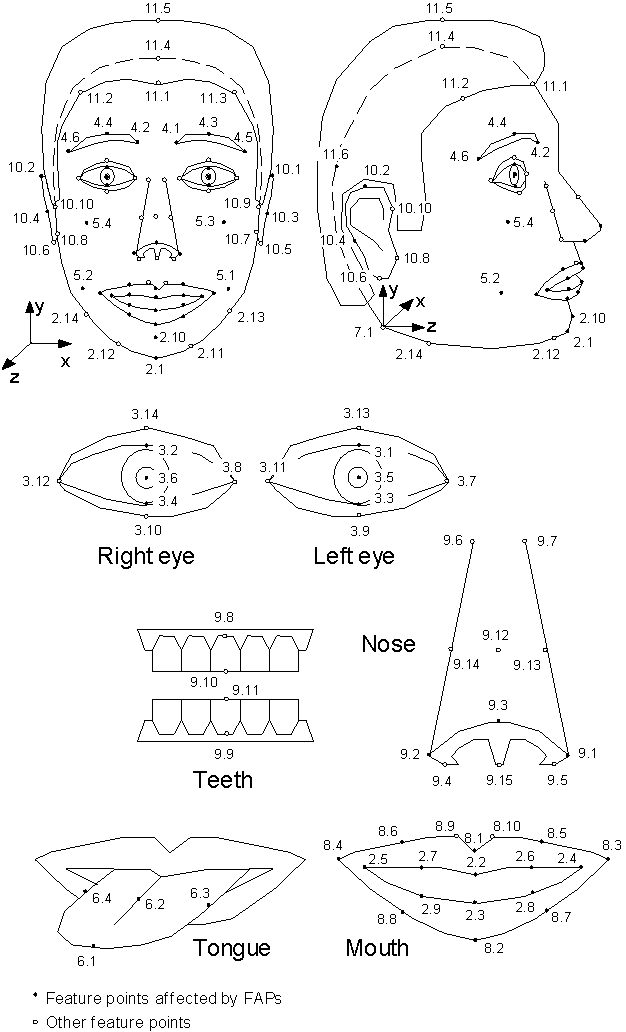


Fig. 15: MPEG-4 Face Feature Points [1].

The 68 **Facial Animation Parameters** are categorized into 10 groups related to parts of the face. For each FAP the standard defines:

* FAPU,
* FAP group,
* The direction of positive motion and whether the motion of the feature point is unidirectional (see FAP 3, open jaw) or bi-directional (see FAP 48, head pitch).

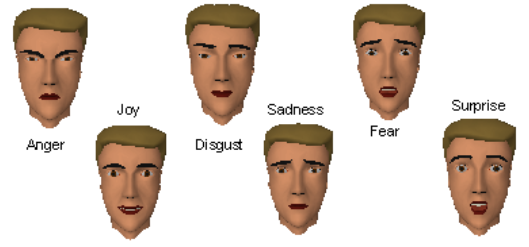


Fig. 16: MPEG-4 expressions [1].

FAPs in the group 1 are high-level animation parameters. A *viseme* (FAP 1) is a visual correlate to a phoneme. Only 14 static visemes that are clearly distinguished are included in the standard set. The FAP 2 defines the 6 primary *facial expressions*.

FAP1 and FAP2 can be applied simultaneously and have an amplitude in the range of [0-63], defined for each expression.

Using FAP 1 and FAP 2 together with low-level FAPs 3-68 that affect the same areas as FAP 1 and 2, may result in unexpected visual artifacts. Generally, the low level FAPs have priority over deformations caused by FAP 1 or 2. When FAPs are recognized automatically as a result of face tracking, only low-level FAPs are coded.

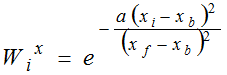
In applications like text-to speech animation is driven only by FAPs 1 and 2, and some periodical motions may be added by designer for head rotations, eyes blinking, breath.

In the Russian department of Intel Corporation (Nizhny Novgorod) MPEG-4 compliant face animation pipeline was implemented. In the lecture it is considered as an example. The pipeline contained the next blocks: creation of head model, modeling of face deformation according to animation parameters in all range, head tracking and FAPs recognition, MPEG-4 coding and decoding of the transferring FAP stream, head model animation and visualization. The blocks of *offline modeling of model deformation* and *online model deformation* according to the set of FAPs are the main point of interest in this lecture.

MPEG-4 defines FAPs, but does not define the implementation of facial animation in decoder. In IFAL (Intel Face Animation Library) for simulation of face deformation for a range of low level FAP values the geometry modeling was applied. For visemes and expressions, the muscle model was implemented.

For each low-level FAP the region of influence was defined in such a way, that 1) the borders were calculated by the coordinates of Feature Points, and 2) no any other FP, to which FAP of the same direction can be applied, was inside the region. While the FAP defines the displacement of feature point along one of the axes, the displacement along other axes was calculated with the constraint to keep the vertex *on the model surface*. According this the vertex-feature point “slides” on the surface from one triangle to another one. The displacement of all the rest vertices in the region was calculated as:

, (1)

, (2)

where *d ri* is the displacement of any vertex in the region, *d rf* is the displacement of feature point, *Wix*is a weight, calculated by coordinates of the vertex, feature point, and the nearest border; *a* is a coefficient, controlling the smoothness of the displacements.

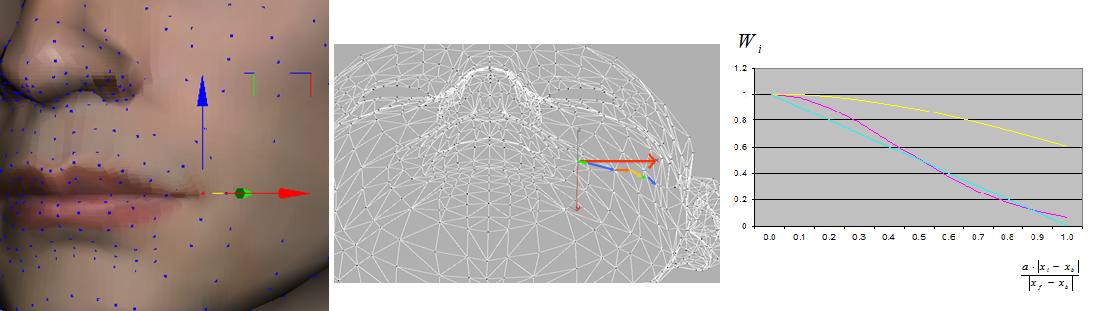


Fig.17: Modeling face deformation by sliding vertices on face surface [2].

The muscle model considered 4 layers: Skin, Fat layer, Muscle layer and Skull. Skin and vertices motion is described with Newton’s low equation as follows:

(3)



Outer forces for vertex *i* in the **fat layer** is a sum of:

* muscle insertion forces

, (4)



* skull reaction force:

, (5)



where the skull reaction force value is opposite to ,



* the force due to muscle volume changing **g***i*.

gi = - *kM* · Δ**d***M*, (6)

where *kM* is the coefficient of the muscle volume changing reaction, Δ**d***M*is the vector of source muscle thickness changing.

For **skin** vertices the sum contains one component – the reaction force **h***i* of skin-fat layer thickness changing:

**h***i* = - *kF* · Δ**d***F*, (7)

*kF* is the coefficient of fat thickness changing, Δ**d***F* is the vector of skin-fat layer thickness changing.

This muscle model allows creating realistic deformation of facial tissues and was applied to get model deformation for visemes and expressions. The deformation was calculated *offline*, and the result according to MPEG-4 data structures was stored in the FaceDefTable – a table, describing face deformation.

During *online process of face animation* the model was deformed on each frame according the set of FAPs values and with use of FaceDefTable information.

To get realistic view of the animated model it is important to have accurate implementation of human skin rendering. Different approaches are presented: from tricky NVIDIA decision, described in [3], to complete realistic skin modeling, described in [4].

As an example of modern state of art face modeling and animation, the technique of creation of the film “Curios Case of Benjamin Button” [5] is discussed. The realistic appearance of Benjamin Button was a result of work of about 100 artists, designers and engineers during 2 year.

**References**

1. Murat Tekalp and Jörn Ostermann. [Face and 2-D Mesh Animation in MPEG-4](http://leonardo.telecomitalialab.com/icjfiles/mpeg-4_si/8-SNHC_visual_paper/8-SNHC_visual_paper.htm) // Signal Processing Image Communication. - 2013. - № 6. DOI: 10.1016/S0923-5965(99)00055-7.
2. A. Fedorov, T. Firsova, V. Kuriakin, E. Martinova, K. Rodyushkin, V. Zhislina. Talking Head: Synthetic Video Facial Animation in MPEG-4 *//* 13-th Int. Conf. on Computer Graphics and Vision GraphiCon’2003. – 2003. - Moscow. - pp. 37-41.
3. Elena Martinova. Realistic Skin Rendering on GPU // Studies in Computational Intelligence. Intelligent Computer Graphics 2009. – 2009. – Volume 240. - Dimitri Plemenos, Georgios Miaoulis (Eds.) – pp. 1-18.
4. Eugene d’Eon. [Advanced Techniques for Realistic Real-Time Skin Rendering](http://http.developer.nvidia.com/GPUGems3/gpugems3_ch14.html) // GPU Gems 3 – 2007. - Chapter 14. - URL: http://http.developer.nvidia.com/GPUGems3/gpugems3\_ch14.html
5. Barbara Flueckiger. [Computer-Generated Characters in Avatar and Benjamin Button](file:///C:\D\doc\Lectures\ComputerAnimation\Материалы%20для%20подготовки\RequiredReading\AvatarButtonFlueckiger.pdf) – 2011. - Translation of Barbara Flueckiger (2011): Zur digitalen Animation von Körpern in Benjamin Button und Avatar. In: Harro Segeberg (ed.): Digitalität und Kino. Munich: Fink. - URL: <http://www.zauberklang.ch/AvatarButtonFlueckiger.pdf>

**Lecture 9. Deformable Bodies.**

*Physically based deformable models* are widely used in computer graphics. The deformable models are active: they respond in a natural way to applied forces, constraints, and impenetrable obstacles. The models are *fundamentally dynamic* and realistic animation is created by numerically solving their underlying differential equations. The methods employ the *theory of elasticity* and may be applied to modeling a set of objects like rubber, cloth, paper, flexible metals and many others.

A deformable object is typically defined by its:

* *undeformed shape* (also called equilibrium configuration, rest or initial shape)
* a set of *material parameters* that define how it deforms under external forces.

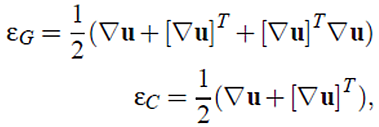
When forces are applied, the object deforms and a point originally at location **m** (i.e. with material coordinates **m**) moves to a new location **x**(**m**), the spatialor world coordinates of that point (see figure 18). The deformation can be specified by the displacementvector field defined on *M*:

**u**(**m**) = **x**(**m**)*−***m**. (1)

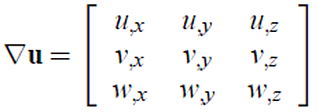
From **u**(**m**) the elastic strain ε is computed: ε is a dimensionless quantity which, in the (linear) 1D case, is simply Δ*l/l*.

The strain must be measured in terms of spatial variations of the displacement field

**u** = **u**(**m**) = (*u, v, w*)*T* . Popular choices in Computer Graphics are

 (2, 3)

where the symmetric tensor ε*G ∈* R3*x*3 is Green’s **nonlinear** strain tensor and ε*C ∈* R3*x*3 its linearization, Cauchy’s **linear** strain tensor. The gradient of the displacement field is a 3 by 3 matrix where the index after the comma represents a spatial derivative:

 (4)

For the computation of the symmetric internal stress tensor σ *∈* R3*x*3 for each material point **m** the strain ε at that point is used.

According to Hooke’s linear material law the stress tensor is

 (5)

where **E** is a rank four tensor, which relates the coefficients of the stress tensor linearly to the coefficients of the strain tensor. For isotropic materials, the coefficients of **E** only depend on Young’s modulus and Poisson’s ratio.

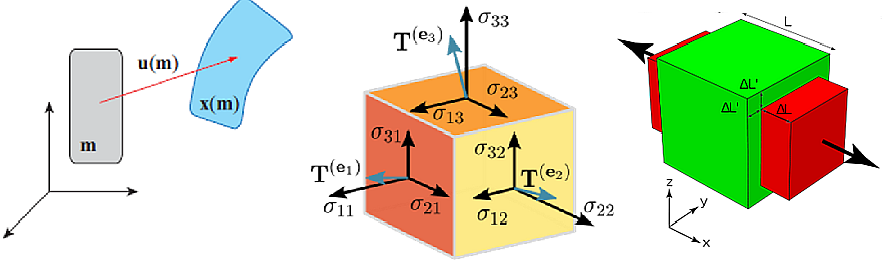


Fig.18: From left to right: Object deformation; Cauchy’s stress tensor; Poisson’s ratio.

The vector field **x**(*t*) is given as a solution of Newton’s second law equation of the form

 (6)

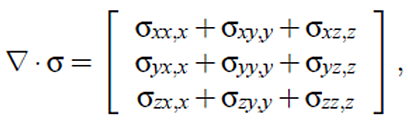
where *F*() is a general function given by physical model of the deformable object. This model can also take into account gravitational force, the force on the surface of a body due to a viscous fluid and other forces, specific for a model (see [1]).

A discrete set of values **x**(0)*,***x**(Δ*t*)*,***x**(2Δ*t*)*, ..* of the unknown vector field **x** which is needed for the animation can now be obtained by numerically solving (i.e. integrating) this system of equations. For this task different methods can be applied, for example the forward-backward Euler scheme (see [5]).

*The Finite Element Method* (FEM) is one of the most popular methods in Computational Sciences to solve Partial Differential Equations (PDE’s) on irregular grids. In order to use the method for the simulation of deformable objects, the object is viewed as a continuous connected volume, which is discretized using an irregular mesh. The PDE describing dynamic elastic materials is given by

 (7)

where ρ is the density of the material and **f** externally applied forces such as gravity or collision forces. The divergence operator turns the 3 by 3 stress tensor back into a 3 vector

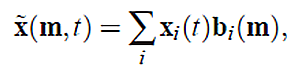
 (8)

representing the internal force resulting from a deformed infinitesimal volume.

The Finite Element Method is used to turn a PDE into a set of algebraic equations, which are then solved numerically.

The domain *M* is discretized into a finite number of disjoint elements (i.e. a mesh). Instead of solving for the spatially continuous function **x**(**m***, t*), it is possible to solve for the discrete set of unknown positions **x***i*(*t*) of the nodes of the mesh.

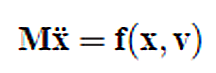
The function **x**(**m***, t*) is approximated using the nodal values by

 (9)

where **b***i*() are fixed nodal basis functions which are 1 at node *i* and 0 at all other nodes, also known as the Kronecker Delta property. Substitution of the last into equation (7) results into algebraic equations for the **x***i*(*t*).

*Mass-spring systems* are arguably the simplest and most intuitive of all deformable models. Instead of beginning with a PDE such as equation 7 and subsequently discretizing in space, the problem is formulated directly with a discrete model. These models simply consist of point masses connected together by a network of massless springs.

The state of the system at a given time *t* is defined by the positions **x***i* and velocities **v***i* of the masses *i* = 1*..n*. The force **f***i* on each mass is computed due to its spring connections with its neighbors, along with external forces such as gravity, friction, etc. The motion of each particle is then governed by Newton’s second law, which for the entire particle system can be expressed as

 (10)

where **M** is a 3*n ×*3*n* diagonal mass matrix. Thus, mass-spring systems require the solution of a system of coupled ordinary differential equations (ODEs).

**References**

1. [Terzopoulos, Demetri, John Platt. Alan Barr, and Kurt Fleischer. Elastically deformable models.](http://design.osu.edu/carlson/history/PDFs/ani-papers/terzopoulos-deformable.pdf) // SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques. - 1987. – pp. 205-214.
2. [Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman and Mark Carlson. Physically Based Deformable Models in Computer Graphics](http://matthias-mueller-fischer.ch/publications/egstar2005.pdf) // Computer Graphics Forum. – 2006. - № 25(4). – pp. 809-836.
3. MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching // ACM Transactions on Computer Graphics. – 2005. -№ 24, 3 – pp. 471-478.
4. David Baraff. [Physically Based Modeling: Principles and Practice Implicit Methods for Differential Equations](https://www.cs.cmu.edu/~baraff/sigcourse/notese.pdf) - Physically Based Modeling: Principles and Practice. (Online Siggraph '97 Course notes). – 1997. - URL: https://www.cs.cmu.edu/~baraff/sigcourse/notese.pdf.

**Lecture 10. Cloth.**

Cloth is a type of *deformable objects*, which was discussed in the previous lecture. The methods of deformable objects simulation can be applied to the cloth animation,

though this problem has some features. The lecture analyses an example implementation of cloth simulation, described in the paper [*Large Steps in Cloth Simulation*](http://run.usc.edu/cs599-s10/cloth/baraff-witkin98.pdf) of David Baraff and Andrew Witkin.

Given a mesh of *n* particles, the position in world-space of the *i*th particle is . The same component notation applies to forces: a force on the cloth exerts a force **f***i* on the *i*th particle. The rest state of cloth is described by assigning each particle an unchanging coordinate (*ui,*,*vi*) in the plane.

The most critical forces in the system are the internal cloth forces, which impart much of the cloth’s characteristic behavior. Breen *et al*. [4] describes the use of the Kawabata system of measurement for realistic determination of the in-plane shearing and out-of-plane bending forces in cloth. These two forces are pointed as the **shear** and **bend** forces. The shear force is formulated on a per triangle basis, while the bend force - on a per edge basis, between pairs of adjacent triangles. The strongest internal force—the **stretch** force— resists in-plane stretching or compression, and is also formulated per triangle.

Complementing the above three internal forces are three **damping** forces. Damping forces that subdue any oscillations having to do with, respectively, stretching, hearing, and bending motions of the cloth. Additional forces include **air**-**drag**, **gravity**, and **user**-**generated** **mouse**-**forces** (for interactive simulations).

Combining all forces into a net force vector **f**, the acceleration of particle is

 (1)

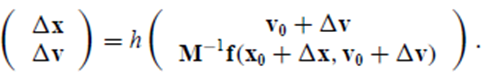
where the mass *mi* is determined by summing one third the mass of all triangles containing the *i*th particle. (A triangle’s mass is the product of the cloth’s density and the triangle’s fixed area in the *uv* coordinate system.)

Defining the diagonal mass matrix **M** by diag(**M) = (***m*1*;m*1*;m*1*;m*2*;m*2*;m*2*; : : : ;mn;mn;mn* ), the equation became

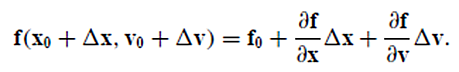
 (2)

To simplify notation, it was defined **x0 =** **x(***t*0)and **v0 =** **v(***t*0), and also define Δ**x =** **x(***t*0 *+* *h*)− **x(***t*0)and Δ**v =** **v(***t*0 *+* *h*)−**v(***t*0).

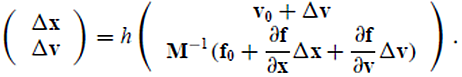
The implicit backward Euler method approximates Δ**x** and Δ**v** by

 (3)

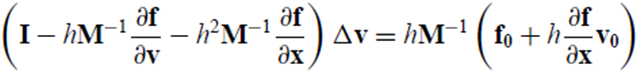
If apply a Taylor series expansion to **f** and make the first order approximation

 (4)

the system became

 (5)

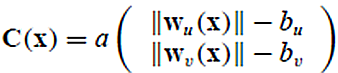
After substitution of the first equation to the second and denote I identity matrix, the final equation is:

 (6)

The equation is solved for Δ**v**, and thenΔ**x** can be calculated.

**Energy and Forces**. Cloth’s material behavior here is described in terms of a scalar potential energy function *E*(**x**); the force **f** arising from this energy is **f** = - 𝜕*E*/𝜕**x**. The internal behavior is defined by formulating a vector condition **C**(**x**)which have to be zero, and then describing the associated energy as *k/*2 **C**(**x**)*T* **C**(**x**), where *k* is a stiffness constant.

**Stretch Forces.** Every cloth particle has a changing position **x***i* in world space, and a fixed plane coordinate (*ui*, *vi)*. It is possible to define a single continuous function **w**(*u*, *v*) that maps from plane coordinates to world space. Stretch can be measured at any point in the cloth surface by examining the derivatives **w***u =* 𝜕**w**/𝜕*u* and **w***v* = 𝜕**w***/*𝜕*v* at that point. The magnitude **w***u* describes the stretch or compression in the *u* direction; the material is unstretched wherever ║**w***u*║= 1. Stretch in the *v* direction is measured by ║**w***v*║. Finally the condition for the stretch energy is

 (7)

where *a* is the triangle’s area in *uv* coordinates. Usually, *bu* = *bv* = 1, but their values can be used as a threshold to produce effect of slight stretching and lengthening a garment.

**Shear Force.** Cloth resists shearing in the plane. The extent to which cloth has sheared in a triangle can be measured by considering the inner product w*u*Tw*v*. In its rest state, this product is zero. By the small angle approximation, the product w*u*Tw*v* is a reasonable approximation to the shear angle.

The condition for shearing is simply

 (8)

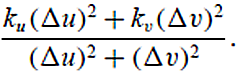
with *a* the triangle’s area in the *uv* plane.

**Bend Force.** Bend is measured between pairs of adjacent triangles. The condition for the bend energy depends upon the four particles defining the two adjoining triangles. If **n**1 and **n**2 denote the unit normals of the two triangles and let **e** be a unit vector parallel to the common edge, the angle *between* the two faces is defined by the relations

sin θ = (n1× n2 )· e and cos θ = n1 · n2. (9)

The condition for bending is simply *C*(**x**)= *θ* which results in a force that counters bending.

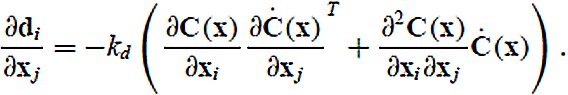
Given material for which bending in the *u* and *v* directions are weighted by stiffnesses *ku* and *kv*, and define for the edge between the triangles between particles *i* and *j*: Δ*u* = *ui* − *uj* and Δ*v* = *vi* − *vj*, the stiffness weighting for this edge is

 (10)

**Damping.** Damping force **d** associated with a condition **C** have the form

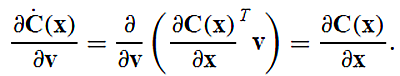
 (11)

Here **d***i* is nonzero only for those particles that **C** depends on. After differentiating this equation, the result is:

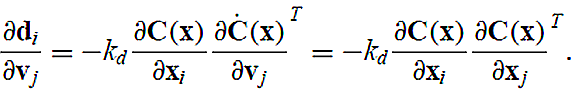
 (12)

Finally, equation (6) requires the derivative 𝜕**d***/*𝜕**v**.

Since **Ċ**(**x**)= (𝜕**C**(**x**)*/*𝜕**x**)*T***v**, the derivative is

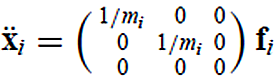
 (13)

And using this fact:

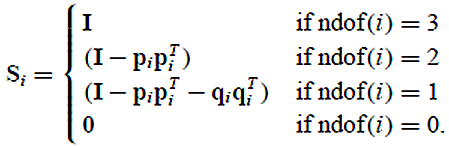
 (14)

**Constraints.** In cloth simulating it is very important to have possibility controlling the particle movement in one, two or three dimensions. A dynamic simulation usually

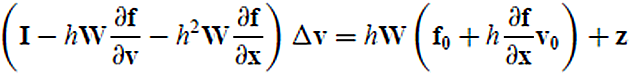
requires knowledge of the *inverse* mass of objects. When inverse mass is used, it becomes trivial to enforce constraints by altering the mass. Suppose for example that it is necessary to keep particle *i*’s velocity from changing. If set 1/*mi* to be zero, the particle mass became infinite mass, making it ignore all forces exerted on it. Complete control over a particle’s acceleration is thus taken care of by storing a value of zero for the particle’s inverse mass. If acceleration must lie in *xy* plane, the inverse mass matrix is used in form:

 (15)

It is possible to define modified version **W** of **M**-1: **W** will be a block-diagonal matrix, which diagonal blocks defined as follows: let ndof(i) indicate the number of degrees of freedom particle *i* has, and prohibited directions be **p***i* (if ndof(i) = 2) or **p***i* and **q***i* (if ndof(i) = 1) with **p***i* and **q***i* mutually orthogonal unit vectors. **W**’s diagonal blocks are **W***ii = 1/mi·***S***i*, where

 (16)

For every particle *i*, let **z***i* be the change in velocity we wish to enforce in the particle’s constrained direction(s). It is possible to point any value of **z***i* for a completely constrained particle, since all directions are constrained; an unconstrained particle must have **z***i* = **0**. Using **W** and **z**, the equation (6) is rewritten to directly enforce constraints:

. (17)

If the equation is solved for Δ**v**, the calculated value is consistent with all constraints.

Completely constrained particles will have Δ**v***i* = **z***i* , while partially constrained particles will have a Δ**v***i* whose component in the constrained direction(s) is equal to **z***i.*

**Colissions**. Given a previous legal state of the cloth, a linear motion for the cloth particles to the current (possibly illegal) state is analyzed and check for either particle/triangle or edge/edge crossings is executed. To avoid *O*(*n*2)comparisons, a coherency-based *bounding box* approach is applied to cull out the majority of pairs.

When collision is detected, a strong damped spring force to push the cloth apart is inserted.

The system detects collisions between cloth particles and *solid objects* by testing each individual cloth particle against the faces of each solid object. A solid object’s faces are grouped in a *hierarchical bounding box tree*, with the leaves of the tree being individual faces of the solid.

After each implicit step, the resulting Δ**x** is analyzed as a proposed change in the cloth’s state, and the stretch terms for each triangle in the newly proposed state are evaluated. If any triangle undergoes a *drastic change in its stretch* (in either the *u* or *v* direction), the proposed state is discarded, the *step size is reduced*, and the step is *repeated* again.

The simulation is run with a parameter (set by the user) that indicates the maximum allowable step size. Whenever the simulator reduces the step size, after two successes with the reduced step size the simulator tries to increase the step size.

If the simulator fails at the larger step size, it reduces the size again, and waits for a longer period of time before retrying to increase the step size. This method, though simple, has served well.

The paper [1] contains more important details about simulation implementation, which can’t be considered in our lecture.

It’s recommended also to look at the tutorials [5], containing many practical recommendation for cloth animation.

The paper, which can also be used to implement cloth simulation is [Deformation Constraints in a MassSpring Model to Describe Rigid Cloth Behavior](http://graphics.stanford.edu/courses/cs468-02-winter/Papers/Rigidcloth.pdf) of Xavier Provot. The forces are represented without energy term. The interesting feature is a research of “Super-Elastic” effect (see the figure below).

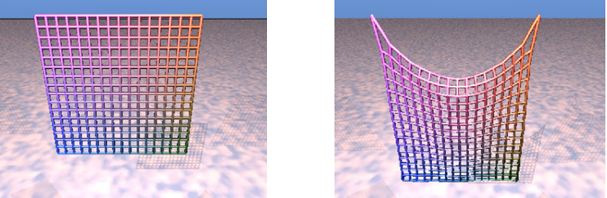


Fig. 19: Initial position and Deformation of the elastic model of a sheet after 200 iterations [2].

**References**

1. David Baraff, Andrew Witkin. [Large Steps in Cloth Simulation](http://run.usc.edu/cs599-s10/cloth/baraff-witkin98.pdf) *//* SIGGRAPH 1998 Proceedings of the 25th annual conference on Computer graphics and interactive techniques – 1998. – pp. 43-54.
2. X. Provot. [Deformation Constraints in a MassSpring Model to Describe Rigid Cloth Behavior](http://graphics.stanford.edu/courses/cs468-02-winter/Papers/Rigidcloth.pdf) // Graphics Interface’95 proceedings – 1995. - A K Peters Ltd, pp 147-154.
3. Jonathan M. Kaldor, Doug L. James, Steve Marschner. [Simulating Knitted Cloth at the Yarn Level](http://www.cs.cornell.edu/~srm/publications/SG08-knit.html) // ACM Transactions on Graphics – 2008. - № 27(3) - pp. 65:1-65:9.
4. D.E. Breen, D.H. House, and M.J. Wozny. [Predicting the drape of woven cloth using interacting particles.](http://accad.osu.edu/~elaine/intrACCAD/cara/cloth/papers/1994-Breen.pdf) // Computer Graphics (Proc. SIGGRAPH) – 1994. - pp 365–372.
5. MAGNENAT-THALMANN N., CORDIER F., KECKEISEN M., KIMMERLE S., KLEIN R., MESETH J. [Simulation of clothes for real-time applications](http://www.cs.jhu.edu/~misha/ReadingSeminar/Papers/Magnenat-Thalmann04.pdf). // In Eurographics 2004, Tutorials 1: Simulation of Clothes for Real-time Applications - 2004. – p. 98.

**Lecture 11-12. Body Animation.**

**Body animation standards.** *MPEG-4* standard contained special sections for virtual actor’s body description and animation, similar to facial animation. The standard defined Body Definition Parameters (BDP): if the source wants the decoder to display a specific body, this one must be sent in the BDP Node. The specific body scene sub-graph replaces the default body model in rendered Body Node. MPEG4 permits also to send *deformation tables* for a proper rendering of the intersecting body parts. To describe a body movement 296 BAP (Body Animation Parameter) can be used. The BAP set parameters defines joint angles for different body parts.

*The Humanoid Animation (H-Anim) Standard* was developed in the late '90s and was the result of research of experts in the graphics, ergonomics, simulation and gaming industry. H-Anim specifies a standard way of representing humanoids in VRML97. Humanoids should work in any VRML97 compliant browser. No assumptions are made about the types of applications that will use humanoids. The human body consists of a number of segments, connected to each other by joints(such as the elbow, wrist and ankle).

Body segments typically are defined by a mesh of polygons, moving according to the skeleton angles. The application may also need to obtain information about which vertices should be treated as a group for the purpose of deformation.

In H-Anim face is described as a set of joints. All facial joints are children of the skullbase joint.

Body animation applications usually are oriented to specific task and thus may use limited set of joints and segments. The common approach is:

* A body is a hierarchical structure, including joints with defined number of DOF;
* While the angles of joint rotation describe posture in the terms of skeleton, the final body appearance must be calculated with use of special algorithm of mesh deformation.

The most popular approach to body deformation is skinning. Mesh vertices are divided to the groups, related to the nearest skeleton segment. During animation vertices from each group has the same matrix as the segment. The vertices between two segments get matrices blended with some weights or calculated with dual quaternions [6]. The resulting deformation does not maintain a constant volume and may cause local self-penetrations.

The work [Implicit Skinning: Real-Time Skin Deformation with Contact Modeling.](http://rodolphe-vaillant.fr/pivotx/templates/projects/implicit_skinning/implicit_skinning.pdf) was presented on SIGGRAPH-2013 [3]. The skinning algorithm based on precomputing *reconstruction* and *composition* (*d* and *e* on the figure 20), and during real-time animation – on projection of vertices in deformed mesh back to the implicit surface.

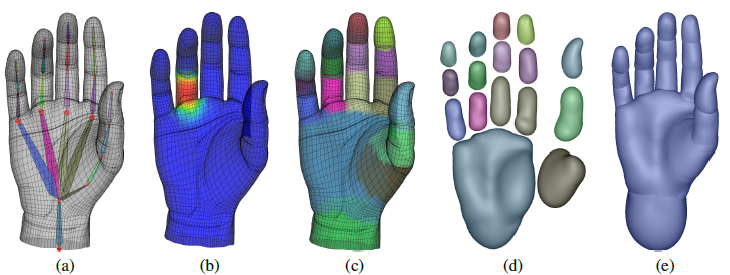


Fig. 20: Implicit skinning [2].(a) An input mesh with its animation skeleton, (b) deformation weights at a joint, and (c) mesh segmentation. (d) Implicit surfaces computed as 0:5-isosurfaces of HRBFs approximating each part of the mesh. (e) Composition with a union operator, and resulting shape.

Reconstruction is executed with use of Hermite Radial Basis Function as a search of

(1)

Interpolating *N* samples (**p***i*, **n***i*) consist in solving for the weights :

(2)

Now >0.5, if **x** is inside, and <0.5, if **x** is outside.

For composition the unit operator is defined:

*f*(**x**)=max(*fi*, *fj*). (3)

This preserve body volume and provide its realistic view in deformation due to the projection step.

In the work of Anguelov et al. [4] statistical analysis of depth scans in different poses is used. Then having motion capture sequence, it is possible to create photorealistic animation and transfer the sequence to the persons with different shapes.

The mesh processing pipeline:

* Get two data sets spanning the shape variability due to different human poses and different physiques.
* Select a few markers by hand mapping the template mesh and each of the range scans.
* Apply the Correlated Correspondence algorithm [4] to compute numerous additional markers.
* Use the markers as input to a non-rigid registration algorithm
* Apply a skeleton reconstruction to recover an articulated skeleton from the registered meshes.
* Learn the space of deformations due to pose and physique.

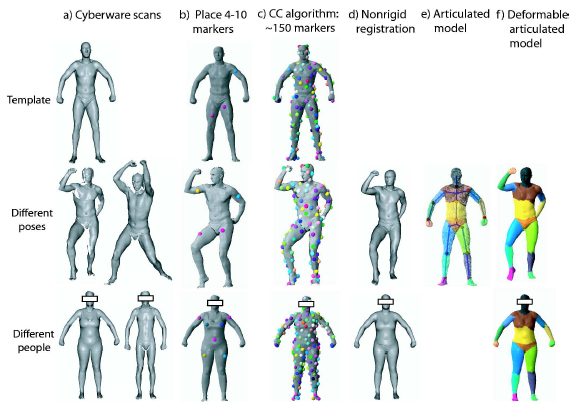


Fig. 21: SCAPE [4]:Animation of a motion capture sequence taken for a subject, having a single body scan. The muscle deformations are synthesized automatically from the space of pose and body shape deformations.

**References**

1. MPEG-4 Specification [ISO/IEC JTC 1/SC 29/WG 11 N2802](http://www.grest.org/data/MPEG4_Spec_Animation.pdf). Information technology – Generic coding of audio-visual objects Part 2: Visual. ISO/IEC 14496-2 FPDAM 1. – 1999. - URL: <http://www.grest.org/data/MPEG4_Spec_Animation.pdf>.
2. H-Anim. [Humanoid Animation Work Group](http://www.h-anim.org/). – URL: <http://www.h-anim.org/>
3. Rodolphe Vaillant Lo¨ıc Barthe, Ga¨el Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, Mathias Paulin. [Implicit Skinning: Real-Time Skin Deformation with Contact Modeling.](http://rodolphe-vaillant.fr/pivotx/templates/projects/implicit_skinning/implicit_skinning.pdf) // ACM Transactions on Graphics (TOG) - SIGGRAPH 2013 Conference Proceedings. – 2013. - № 32, 4. - Article No. 125. – p.11.
4. Dragomir Anguelov , Praveen Srinivasan , Daphne Koller , Sebastian Thrun , Jim Rodgers , James Davis. SCAPE: shape completion and animation of people // SIGGRAPH '05 ACM SIGGRAPH 2005 Papers. – 2005. – pp. 408-416.
5. KRY P. G., JAMES D. L., PAI D. K. Eigenskin: real time large deformation character skinning in hardware // In Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation. –2002. - pp. 153–159.
6. JAMES D. L., TWIGG C. D. Skinning mesh animations // ACM Transactions on Graphics (SIGGRAPH 2005). - 2005. - № 24, 3. – pp. 399-407.

**Lecture 13-14. Fluids. Smoke. Fire.**

Physically based animation of fluids such as smoke, water, and fire historically been the domain of high-quality offline rendering due to great computational cost. Currently on modern GPU it is possible not only simulate and render fluids in real time, but also they can be seamlessly integrated into real-time applications. The task remains challenge not only because fluids are expensive to simulate, but also because the volumetric data produced by simulation does not fit easily into the standard rasterization-based rendering paradigm. The lecture based mainly on ideas of GPU Gems papers [1],[2] with use of details from cited sources [3-6].

**Navier-Stokes Equations.**

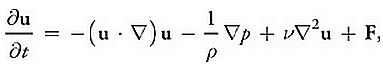
The most important quantity to represent the behavior of a fluid is the velocity of the fluid. The fluid's velocity varies in both time and space and represents a *vector field*.

A vector field is a mapping of a vector-valued function onto a parameterized space, such as a Cartesian grid. (Other spatial parameterizations are possible). The velocity vector field of fluid is defined such that for every position **x** = (*x*, *y, z*), there is an associated velocity at time *t*, **u**(**x**, *t*) = (*u*(**x**, *t*), *v*(**x**, *t*), *w*(**x**,*t*)) as shown in Figure.

In fluid dynamics it's common to assume an *incompressible, homogeneous* fluid.

A fluid is incompressible if the volume of any subregion of the fluid is constant over time. A fluid is homogeneous if its density, *r*, is constant in space. These assumptions do not decrease the applicability of the resulting mathematics to the simulation of real fluids, such as water and air.

Fluid dynamics is simulated on a regular Cartesian grid with spatial coordinates **x** = (*x*, *y, z*) and time variable *t*. The fluid is represented by its velocity field **u**(**x**, *t*) and a scalar pressure field *p*(**x**, *t*). If the velocity and pressure are known for the initial time *t* = 0, then the state of the fluid over time can be described by the *Navier-Stokes equations for incompressible flow*:

 (1)

 (2)

where *ρ* is the (constant) fluid density, *ν* is the kinematic viscosity, and **F** = (*fx* , *fy, fz*) represents any external forces that act on the fluid. Notice that Equation 1 is actually three equations, because **u** is a vector quantity.

*Advection*. The velocity of a fluid causes the fluid to transport objects, densities, and other quantities along with the flow. The objects are transported, or *advected*, along the fluid's velocity field. The first term on the right-hand side of Equation 1 represents this *self-advection* of the velocity field and is called the *advection term*.

*Pressure*. When force is applied to a fluid, the molecules close to the force push on those farther away, and pressure builds up. Because pressure is force per unit area, any pressure in the fluid naturally leads to acceleration. The second term, called the *pressure term*, represents this acceleration.

*Viscosity* is a measure of how resistive a fluid is to flow. This resistance results in diffusion of the momentum (and therefore velocity), so the third term is the *diffusion* term.

The fourth term encapsulates acceleration due to *external forces* applied to the fluid. These forces may be either *local forces* or *body forces*.

*The Helmholtz-Hodge Decomposition*. Let *D* be the region in space, on which the fluid is defined. Let this region have a smooth (that is, differentiable) boundary, U2202.GIF*D*, with normal direction **n**. According to *The Helmholtz-Hodge Theorem,* a vector field **w** on*D*can be uniquely decomposed in the form:

** (3)

where **u** has zero divergence and is parallel to U2202.GIF*D*; that is, **u** · **n** = 0 on U2202.GIF*D*. This theorem states that any vector field can be decomposed into the sum of two other vector fields: a divergence-free vector field, and the gradient of a scalar field.

Solving the Navier-Stokes equations involves three computations to update the velocity at each time step: advection, diffusion, and force application. The result is a new velocity field, **w**, with *nonzero* divergence. But the continuity equation (2) requires a divergence-free velocity at the end of each time step. Fortunately, the Helmholtz-Hodge Decomposition Theorem allows the divergence of the velocity to be corrected by subtracting the gradient of the resulting pressure field:

 (4)

The theorem also leads to a method for computing the pressure field. If we apply the divergence operator to both sides of Equation 3, we obtain:

 (5)

Since Equation 2 enforces that  this simplifies to:

 (6)

which is a Poisson equation for the pressure of the fluid. This means that after divergent velocity, **w**, was calculated, it is possible to solve Equation 6 for *p*, and then use **w** and *p* to compute the new divergence-free field, **u**, using Equation 4.

From the definition of the dot product, the projection of a vector r onto a unit vector can be found by computing the dot product of r and . The dot product is a *projection operator* for vectors that maps a vector **r** onto its component in the direction of . The Helmholtz-Hodge Decomposition Theorem can be used to define a projection operator,  , that projects a vector field **w** onto its **divergence-free component, u.** If apply  to Equation 3, we get:

 (7)

But by the definition of  ,

 (8)

Therefore, 

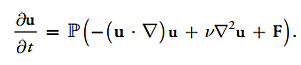
These ideas can be used to simplify the Navier-Stokes equations. If apply the projection operator to both sides of Equation 1:

 (9)

Because **u** is divergence-free, so it’s derivative on the left-hand side

 (10)

Since  the pressure term drops out. Now equation is:

 (11)

This equation encapsulates entire algorithm for simulating fluid flow. From left to right, we compute all values inside the parentheses: advection, diffusion, and force terms. This results in a divergent velocity field, **w**, to which the projection operator is applied to get a new divergence-free field, **u**. To do so, the Equation 6 is solved for p, and then subtract the gradient of p from w, as in Equation 4.

In a typical implementation, the solution is found via composition of transformations on the state: each component is a step that takes a field as input, and produces a new field as output.

The operator  is defined that is equivalent to the solution of Equation 11 over a single time step: it is the composition of operators for advection (), diffusion (), force application (), and projection ():

 (12)

The operators are applied right to left: advection, diffusion, force application, projection.

To compute the *advection* of a quantity, we must update the quantity at each grid point. Taking into account, that use explicit methods for advection are unstable for large time steps, and GPU specific, the next equation is used to update a quantity *q* (this could be velocity, density, temperature, or any quantity carried by the fluid):

 (13)

A partial differential equation for *viscous diffusion* is:

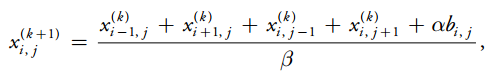
 (14)

The implicit formulation, which is stable for arbitrary time steps and viscosities:

 (15)

where I is the identity matrix. This equation is a *Poisson equation for velocity*. Remember that the use of the Helmholtz-Hodge decomposition results *in a Poisson equation for pressure*. These equations can be solved using an iterative relaxation technique.

Equations 10 and 15 appear different, but both can be discretized using finite difference form for Laplacian and rewritten in the form:

 (16)

where *α* and β are constants. The values of *x*, *b*, *α*, and *β* are different for the two equations. In the Poisson-pressure equation, *x* represents *p*, *b* represents ∇ ⋅ w, 

For the viscous diffusion equation, both *x* and *b* represent **u**, 

This formulation of the equations lets to use the same code to solve either equation.

To solve the equations, we run a number of Jacobi iterations in which we apply Equation 16 at every grid cell, using the results of the previous iteration as input to the next (*x*(*k+1*) becomes *x*(*k*) ).

As an initial condition, it is assumed the fluid initially has zero velocity and zero pressure everywhere. For velocity, it is applied the *no-slip* condition, which specifies that velocity goes to zero at the boundaries. The correct solution of the Poisson-pressure equation requires *pure Neumann* boundary conditions: ∂p/∂n = 0. This means that at a boundary, the rate of change of pressure in the direction normal to the boundary is zero.

For simulation an *Eulerian* discretization with fixed in space computational elements is used. The rectilinear volume is subdivided into a regular grid of cubical cells.

Each grid cell stores both scalar quantities (such as pressure, temperature, and so on) and vector quantities (such as velocity). This scheme makes implementation on the GPU simple, because there is a straightforward mapping between grid cells and voxels in a 3D texture.

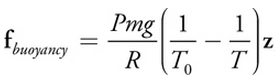
To discretize *derivatives* in the equations *finite differences* numerically approximate derivatives by taking linear combinations of values defined on the grid. In a GPU implementation, cell attributes (velocity, pressure, and so on) are stored in several 3D textures.

At each simulation step, these values are updated by running computational *kernels* over the grid. A kernel is implemented as a pixel shader that executes on every cell in the grid and writes the results to an output texture.

Common equations of fluid simulation has some specific features, when applied to smoke, fire and liquids.

For instance, to obtain the appearance of **smoke** it is possible to keep track of density and temperature. For each additional quantity one must allocate an additional texture with the same dimensions as the grid. The evolution of values in this texture is governed by the same advection equation used for velocity.

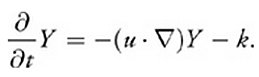
The temperature values have an influence on the dynamics of the fluid. This influence is described by the *buoyant force*:

 (17)

where *P* is pressure, *m* is the molar mass of the gas, *g* is the acceleration due to gravity, and *R* is the universal gas constant. The value *T* 0 is the ambient or "room" temperature, and *T* represents the temperature values being advected through the flow. **z** is the normalized upward-direction vector.

The buoyant force should be thought of as an "external" force and should be added to the velocity field immediately following velocity advection.

**Fire** is not very different from smoke except that an additional quantity, called the reaction coordinate, is stored. A reaction coordinate of one indicates that the gas was just ignited, and a coordinate of less than zero indicates that the fuel has been completely exhausted. The evolution of these values is described by the following equation:

 (18)

The reaction coordinate is advected through the flow and decremented by a constant amount (*k*) at each time step.

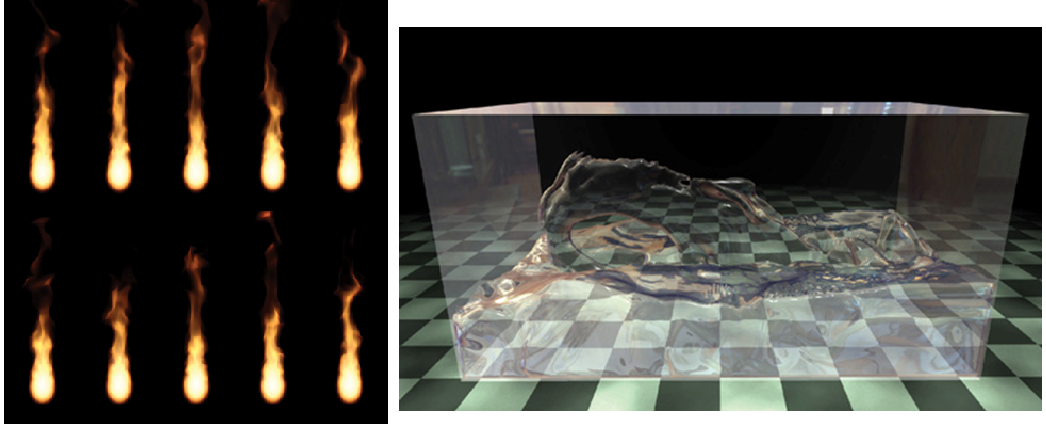


Fig. 22: Simulation a ball of fuel by setting the reaction coordinate in a spherical region to one; water simulation result with refraction [2].

Reaction coordinates do not have an effect on the dynamics of the fluid but are later used for rendering. The figure 22 demonstrates one possible fire effect: a ball of fuel is continuously generated near the bottom of the volume by setting the reaction coordinate in a spherical region to one. For a more advanced treatment of flames, see Nguyen et al. 2002 [4].

**Water** is modeled differently from smoke and fire: the visually interesting part is the *interface* between air and liquid.

The *level set method* is a popular representation of a liquid surface and is particularly well suited to a GPU implementation because it requires only a scalar value at each grid cell. Each cell records the *shortest signed distance* *h* from the cell center to the water surface. Cells in the grid are classified according to the value of *h*: if  *h* < 0, the cell contains water; otherwise, it contains air.

Wherever *h* equals zero is exactly where the water meets the air (the *zero set*).

Because advection will not preserve the distance field property of a level set, it is common to periodically *reinitialize* the level set. Reinitialization ensures that each cell does indeed store the shortest distance to the zero set. In fact, the level set *defines* the fluid domain.

In practice, the pressure outside of the liquid is set to zero before solving for pressure and modify the pressure only in liquid cells. It also means that the external forces such as gravity aren’t applied outside of the liquid.

**Rendering**. The result of fluid simulation is a collection of values stored in a 3D texture and used to render realistic scene with smoke, fuel or water.

To render the fluid a *ray-marching* pixel shader, for example, may be used. The approach is similar to the one described in the [5]: Scharsach, H. 2005. "Advanced GPU Raycasting."

The placement of the fluid in the scene is determined by six quads, which represent the faces of the simulation volume. These quads are drawn into a deferred shading buffer to determine where and how rays should be cast. The fluid is rendered by marching rays through the volume and accumulating densities from the 3D texture, as shown below.

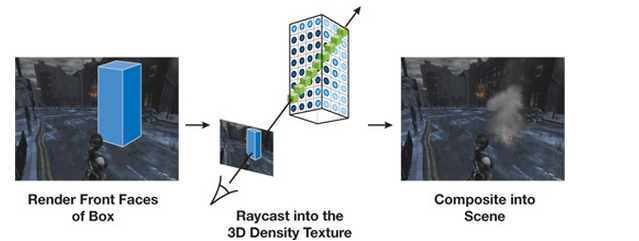


Fig. 23: Process of smoke rendering [2].

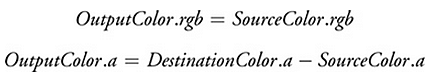
For each ray it is possible to calculate:

* where it enters the volume,
* the direction in which it is traveling,
* how many samples to take (for example, 2 samples per each voxel).

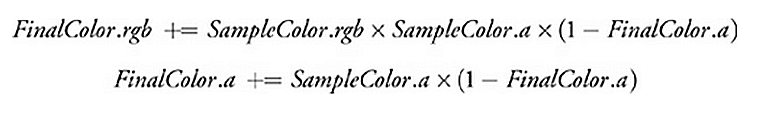
This information is precalculated and stored in the *RayData* texture, which encodes, for every pixel that is to be rendered:

* the entry point of the ray, and
* the depth through the volume that the ray traverses.

To get the depth through the volume, first the back faces of the volume are drawn with a shader that outputs the distance from the eye to the fragment's position (in view space) into the alpha channel. Then a similar shader is called on the front faces with enabling subtractive blending using equations below. To get the entry point of the ray, the texture-space coordinates of each fragment (i.e. coordinates in the simulation volume) generated for the front faces are also output into the RGB channel.

 (19, 20)

Then a ray-marching shader looks up into the *RayData* texture to find the ray entry point and marching distance through the volume for the pixels along the ray. The ray direction is given by the vector from the eye to the entry point. At each step along the ray, the values from the texture containing the simulated values are used and blended *front to back* according to equations below.

 (21, 22)

By blending from front to back, it is possible to terminate ray casting early if the color saturates (for example, if *FinalColor.a* > 0.99).

**Rendering fire** is similar to rendering smoke except that instead of the smoke density the reaction coordinate  *Y* determines the blending values. In particular, it is possible to use an artist-defined 1D texture that maps reaction coordinates to colors in a way that gives the appearance of fire.

The fire volume can also be used as a *light source*. The simplest approach is to sample at several locations and treat each sample as a point light source. However, this approach can lead to severe flickering if not enough point samples are used, and it may not capture the overall behavior of the light.

A different approach is to downsample simulation volume of reaction coordinates to an extremely low resolution and then use *every* voxel as a light source. The latter approach will be less prone to flickering, but it won't capture any high-frequency lighting effects (such as local lighting due to sparks).

To render a **liquid surface**, it is necessary to march through a volume, but this time the values depends on the level set. For water, it is particularly important that no artifacts of the grid resolution are seen because it is possible to use tricubic interpolation to filter these values.

For fluids like water, there are several ways to make the surface appear as though it *refracts the objects* behind it. *Ray tracing* is expensive, and there may be no way to find ray intersections with other scene geometry.

It is possible to use an approximation that gives the impression of refraction but is fast and simple to implement:

* First, the objects behind the fluid volume are rendered into a background texture.
* Next, the nearest ray intersection with the water surface at every pixel is determined by marching through the volume. This produces a pair of hit locations and shading normal. If there was ray-surface intersection at a pixel, it is shaded with a refraction shader that uses the background texture. Finally, foreground objects are added to create the final image.

The appearance of refraction is achieved by looking up a pixel in the background image near the point being shaded and taking its value as the refracted color. This background pixel is accessed at a texture coordinate **t** that is equal to the location **p** of the pixel being shaded with the offset to a vector proportional to the projection of the surface normal **N** onto the image plane:

 (23)

where Ph and Pv are an orthonormal basis for the image plane and are defined as:

 (24)

 (25)

where **z** is up and **V** is the view direction.

The effect of applying this transformation to the texture coordinates is that a convex region of the surface will magnify the image behind it, a concave region will shrink the image, and flat (with respect to the viewer) regions will allow rays to pass straight through.

**References**

1. Mark J. Harris. Fast Fluid Dynamics Simulation on the GPU // GPU Gems. -2004. – URL: <https://developer.nvidia.com/gpugems/GPUGems/gpugems_chapter38.html>.
2. Keenan Crane, Ignacio Llamas, Sarah Tariq. Real-Time Simulation and Rendering of 3D Fluids // GPU Gems-3. – 2007. – URL: <https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch30.html>
3. Bridson R., M. Muller-Fischer. Fluid Simulation: SIGGRAPH 2007 Course Notes // In Proceeding SIGGRAPH ‘07 ACM SIGGRAPH 2007 courses – 2007. – pp. 1-81.
4. Nguyen, D., R. Fedkiw, and H. W. Jensen. Physically Based Modeling and Animation of Fire // In ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002). - 2002. - № 21(3). – pp. 721 – 728.
5. Scharsach, H. Advanced GPU Raycasting // In Proceedings of CESCG. – 2005. – pp. 69 – 79.
6. Fedkiw, R., J. Stam, and H. W. Jensen. 2001. Visual Simulation of Smoke // In  SIGGRAPH ‘ 2001 Proceedings of the 28th annual conference on Computer graphics and interactive techniques. – 2001. - pp. 15–22.

**Lecture 15. 3D Animation Software.**

*3D modeling software* is a class of *3D computer graphics software* used to produce 3D models. Individual programs of this class are called modeling applications or modelers. Some of them have **animation features**.

The list (incomplete) of **3D Animation software** can be found here: <http://en.wikipedia.org/wiki/List_of_3D_animation_software>.

3D modelers can export their models to files in the different format and also import files from other applications.

**Autodesk 3ds Max** offers powerful capabilities for creating professional-quality 3D animations, renders, and models. It has an efficient creative toolset with a with long history of development and rich traditions. Overview of capabilities can be found here: <http://www.autodesk.ru/products/3ds-max/overview>.

Autodesk 3ds Max, formerly **3D Studio Max**, is a professional 3D computer graphics program for making 3D animations, models, games and images. It is developed and produced by Autodesk Media and Entertainment.

It has modeling capabilities, a flexible plugin architecture and can be used only on the Microsoft Windows platform.

* Current version: 2016 / 12 April 2015.
* Operation System: Windows 7 – Windows 8.1
* Platform:x64, x86 (since 2014 only x64)
* Website: <http://www.autodesk.com/3dsmax>
* System requirements: 64-bit Intel® or AMD® multi-core processor, 4 GB of RAM (8 GB recommended), 6 GB of free disk space for install, graphics hardware – from the approved list, for example NVIDIA Quadro, Fermi or Kepler, from 1024 MB. It is important to mention, that for really good performance and comfortable use it is necessary to have at least twice more video memory.

3ds provides a complete set of 2D and 3D modeling and texturing:

* Mesh and surface modeling: Efficiently create parametric and organic objects.
* Polygon, spline, and NURBS-based modeling: Use spline and 2D modeling tools.
* Point cloud support: Create models from point cloud data.
* Vector map support: Load vector graphics as texture maps.
* Texture assignment and editing: Explore an advanced texturing toolset.
* ShaderFX: Intuitively create advanced HLSL shaders
* Enhanced ShaderFX: Intuitively create and exchange advanced HLSL shaders

Rendering in Autodesk 3ds Max:

* Exposure lighting simulation and analysis: Simulate and analyze sun, sky, and artificial light.
* Render in the cloud right from within 3ds Max.
* Support for new Iray and mental ray enhancements: Rendering photorealistic images
* Integrated rendering options: Achieve stunning image quality with NVIDIA Iray.
* Stereo Camera: Create engaging 3D content.

Animation functions:

* General animation tools: Work with keyframe, Dopesheet, and procedural tools.
* Animated deformers: Add life to creatures and simulate fluidic effects.
* Character animation and rigging tools:Create believable characters with realistic motion.
* Dual Quaternion skinning
* Populate crowd animation: Generate believable human motion.

There are two ways to use 3ds Max for free:

* Free 30 days trial for Windows 64 bit is available here: <http://www.autodesk.com/products/3ds-max/free-trial>
* Free 3 years subscription for students can be obtained here <http://www.autodesk.com/education/free-software/all>

**Autodesk Maya** is a 3D computer graphics software that runs not only on Windows, but also on OS X and Linux.

* Originally developed by Alias Systems Corporation (formerly Alias|Wavefront) and currently owned and developed by Autodesk, Inc. It is used to create interactive 3D applications, including video games, animated film, TV series, or visual effects.
* Current version: 2015 / April 15, 2014.
* Platform:IA-32, x64.
* Website: <http://www.autodesk.com/maya>
* System requirements: 64-bit Intel® or AMD® multi-core processor, 4 GB of RAM (8 GB recommended), 6 GB of free disk space for install, graphics hardware – from the approved list, for example NVIDIA Quadro, Fermi or Kepler, from 1024 MB.

Maya® 3D animation, modeling, simulation, and rendering software offers artists a comprehensive creative toolset. These tools provide a starting point to realize your vision in modeling, animation, lighting, and visual effects (VFX). Overview of capabilities can be found here: <http://www.autodesk.com/products/maya/overview>.

* Generate curves, spheres, and custom geometry.
* Rigid and soft-body dynamics: Simulate multiple rigid and flexible objects.
* Fluid Effects: Simulate atmospherics, liquids, and open water.
* Maya nCloth: Create realistic deformable materials.
* Maya Fur: Create realistic fur, short hair, wool, and grass.
* Bullet Physics: Create realistic rigid and soft-body simulations.
* Maya nHair: Create hair and curve-based dynamics.
* Maya nParticles: Simulate complex 3D visual effects.

Autodesk Maya *Animation:*

* General animation tools: Keyframe, procedural, and scripted animation tools.
* Reusable animation: Reuse existing characters to save time.
* Natural-looking character creation: Skin, rig, and pose believable characters.
* Camera Sequencer: Speed previsualization and virtual moviemaking.
* Geodesic Voxel Binding: Get high-quality skinning results in short time.

**3ds Max vs Maya.** For character animation, Maya may be the best choice. However, 3ds Max still has great animation capabilities, but Maya has a deeper list of tools.

For modeling, either software is going to get the job done. 3ds Max has a robust modeling toolset, but Maya has recently enhanced their tools as well.

3ds Max has typically been seen as the 3D app for the game industry, and it is known to have a bit more flexibility and options; however Maya LT is also a great cost effective choice when it comes to game development.

If you’re going to be doing a lot of architectural visualization then 3ds Max is probably going to be your best bet. Of course, you can do architectural work in Maya, but 3ds Max Design is integrated with some of the other design software like [AutoCAD](http://www.autodesk.com/products/autodesk-autocad/overview).

Finally, Operation System is the main factor for choice.

**Blender** is a professional free and open-source 3D computer graphics software product used for creating animated films, visual effects, art, 3D printed models, interactive 3D applications and video games. Quick overview of features is here: <https://www.youtube.com/watch?v=1XZGulDxz9o&feature=youtu.be>.

* Developer: Blender Foundation.
* Current version: 2.74 / March 31, 2015.
* Operation System: Microsoft Windows, Mac OS X,Linux, FreeBSD
* License: GNU General Public License
* Platform:x64, x86 (since 2014 only x64).
* Website: <http://blender.org/>
* System requirements: 64-bit quad core CPU, 2 GB of RAM (8 GB recommended), 6 GB of free disk space for install, graphics hardware – OpenGL card with 1 GB video RAM (CUDA or OpenCL for GPU rendering).

*Some Features:*

* Photorealistic Rendering
* Fast Modeling: has a rich array of modeling tools make creating, transforming and editing models
* Realistic Materials:
  + Physically accurate shaders like glass, translucency and SSS
  + Open Shading Language (OSL) support for coding unique shaders
* Game Creation: Included in Blender is a complete game engine, allowing you to create a fully featured 3d game right inside Blender.
* Blender comes loaded with a vast array of extensions that you can turn on or off easily.
* Has a built-in Video Editor.
* File Formats: has with import/export support for many different programs and formats of images, video. For 3D:
  + 3D Studio (3DS), COLLADA (DAE), Filmbox (FBX), Autodesk (DXF), Wavefront (OBJ), DirectX (x), Lightwave (LWO), Motion Capture (BVH), SVG, Stanford PLY, STL, VRML, VRML97, X3D

*Animation in Blender***.**

* Fast Rigging: Blender offers a set of rigging tools including:
  + Envelope, skeleton and automatic skinning
  + Easy weight painting
  + Mirror functionality
  + Bone layers and colored groups for organization
  + B-spline interpolated bones
* Blender’s animation feature set offers:
* Automated walk-cycles along paths
  + Character animation pose editor
  + Non Linear Animation (NLA) for independent movements
  + IK forward/inverse kinematics for fast poses
  + Sound synchronization
* Simulations of fluids, smoke, hair, cloth, Rigid Body Physics, particles.

**DAZ Studio** is a 3D figure illustration/animation application. It is compatible with most files intended for use by Poser (see below). It is available for free but registration is required.

* Developer: DAZ 3D.
* Current version: 4.7.0.12 / 18 November 2014.
* Operation System: Windows XP or later, Mac OS X Leopard or later
* License: Professional edition: Freeware
* Platform:IA-32 and x86-64
* Website: <http://www.daz3d.com/studio/>

Features: is designed to allow users to manipulate "ready to use" models and figures. It is aimed at users who are interested in posing human and non-human figures for illustrations and animation, but who do not wish to incur the expense—in terms of time and money—of higher-end 3D and CAD software.

**Poser** is a 3D computer graphics program optimized for 3D modeling of human figures.

* Developer: Smith Micro Software
* Current version: Pro 2014 / May 2013
* Operation System: Windows, OS X
* License: [Trialware](http://en.wikipedia.org/wiki/Trialware)
* Platform:IA-32 and x86-64
* Website: <http://my.smithmicro.com/poser-3d-animation-software.html>
* Price: ~500$

Poser Pro 2014 includes robust 3D character creation tools including clothing fitting, morph target creation, weight mapping tools, network rendering and the full collection of Poser 10 features, but in a native 64-bit application. The included set of PoserFusion 2014 plug-ins are perfect for content integration with Lightwave, CINEMA 4D, 3ds Max and Autodesk Maya, as well as Z-brush via our Go-Z support.

*Exclusive Pro 2014 ONLY features:*

* Fitting Room
* Copy Morphs from Figure to Figure
* Weight Map Creation Tool Suite: Paint weight maps on any joint for super smooth and controlled bending
* HDRI: For photorealistic texture creation, High Dynamic Range (HDRI) images are supported with full depth brightness and color.
* To transport Poser into a variety of powerful third party tools including Sketch-up, Modo and many other powerful pro-tools, Poser has one of the most robust COLLADA import/export pipelines available today.
* PoserFusion Plug-ins : PoserFusion 2014 plug-ins work with Lightwave, CINEMA 4D, Max and Maya, and allow you to import full Poser scenes including character rigging, textures and full dynamics.

**Find more here:**

<http://en.wikipedia.org/wiki/List_of_3D_animation_software>.

**Section II. Main Topics of Practice**

1. Walt Disney Studio principles of tradition animation. Examples of squash and stretch, timing and motion, anticipation, staging, follow through and overlapping action, straight ahead and pose-to-pose actions, slow in and out, appeal, exaggeration, secondary action, arcs usage. Example movies. Practical implementation of at least two principles by every student.
2. Keyframing in motion, shape and color transformations. Linear and spline interpolation, cubic splines. Cubic Hermite spline.Practical implementation of several examples, usage of at least two kinds of interpolation.
3. Motion capture data structure. Review of free sources of motion capture data. Rendering of motion capture data. Motion analysis and synthesis of new sequences.
4. Basic particle system model. Particle generation. Initial attributes and dynamic. Particle extinction. Rendering. Particle hierarchy. Example program implementation, group work to improve appearance of particle system. Development of applications on base of particle systems.
5. VRML. Study of VRML tutorial. VRML players. Work with examples of VRML models. Coding of animation in VRML.
6. Independent work on projects. Topic choice and discussion. 3 stages of discussion: Short abstracts presentation, Project progress report, Final project presentation.

**Section III. Examination Questions in Computer Animation**

1. Animation principles. Classification. Description of principles. Examples.
2. Keyframing. Types of transformation. The ways of key frames creation. The methods of inbetween calculation. Interpolation. Blend shapes. Examples.
3. Motion capture types. Motion capture data use. Motion analysis and synthesis. Motion graph. Physical simulation and motion capture.
4. Particle system definition, features, advantages. Basic Particle system model. Particle Generation. Initial attributes and dynamic. Particle extinction. Particle hierarchy. Rendering. Examples.
5. Rigid body definition and features. Rigid body translation and rotation. Collision Detection. Collision Reaction. The ways to optimize collision simulation.
6. Locomotion Terminology. Gaits description. Legged Locomotion Simulation. Analytical Inverse Kinematics.
7. Face animation in MPEG-4. Parameters in MPEG-4 animation: FDP, FP, FAP, FAPU. High level and low-level animation parameters. Visemes, expressions. Function of FaceDefTable and FIT. MPEG-4 role in animation development. MPEG-4 problems and limitations.
8. Face animation framework. Algorithms of head model deformation. Text-to-speech.
9. Realistic skin rendering. Light-skin interaction. Skin reflectance model. Subsurface scattering. Real time skin rendering.
10. Cloth simulation. Main approach. Forces. Collision detection. Simulation step adaptation.
11. Deformable objects in animation. Dynamic model of object deformation. Numerical integration of equations. The Finite Element Method. Mass-Spring Systems for deformable object simulation.
12. Articulated figures. Human body representation. Body animation standards. Kinematics equations of the serial chain.
13. Body animation: common approach. Skinning and traditional problems. Implicit skinning.
14. Body animation: Statistical approach. SCAPE: Shape Completion and Animation of People.
15. Fluids simulation. Smoke, fire, liquid.
16. Rendering of the results of fluids simulation. Rendering smoke, fire, liquid.
17. 3D modeling software. 3D animation software. Examples of commercial and free tools. Animation functions in 3D animation software.

**Section IV. Problems for independent work.**

1. **Animation Principles**

The task: to develop application, demonstrating 12 Pixar’s animation principles.

Simplified version: to develop application, demonstrating some of animation principles.

1. **Keyframing**

The task: to develop application, demonstrating Keyframing technique with different transforming parameters and algorithms of interpolation.

1. **Motion Graph from open motion capture sources**

The task: to develop application, demonstrating technique of Motion Graph creation and use.

Simplified version: prepare review of free motion capture data sources.

1. **Particle Systems**

The task: to develop application, demonstrating use of particle system.

Simplified version: the application can demonstrate abstract particle system with attractive appearance.

1. **Particle Systems to Model Grass**

The task: to develop application of particle system, modelling grass growth.

1. **Text-to-speech application**

The task: to develop text-to-speech application.

1. **Rigid bodies animation**

The task: to develop application, demonstrating interaction of rigid bodies.

1. **Free topic**

The task: to develop any animation application, related to the computer animation course. Use studied methods or independently find some new approach or idea.

**COMPUTER  
ANIMATION**

Author and editor Elena Martynova

***Studying-methodological manual***

Federal State budget educational institution of higher professional education  
«N.I. Lobachevsky State University of Nizhny Novgorod».

603950, Nizhny Novgorod, Gagarin av., 23.

Sent to the press \_\_.\_\_. 2015. Format 60х84 1/16.

Paper offset. Press offset. Set Times.

Conditional quire \_\_\_. Stud.-publ. lit. \_\_\_.

Order № \_\_\_. Circulation \_\_\_ cop.

Printed in printing house of N.I. Lobachevsky  
State University of Nizhny Novgorod

603600, Nizhny Novgorod, Bol'shaya Pokrovskaya st., 37

License ПД № \_\_-\_\_\_\_ from \_\_.\_\_.\_\_

**КОМПЬЮТЕРНАЯ  
АНИМАЦИЯ**

Автор и составитель Елена Михайловна **Мартынова**

***Учебно-методическое пособие***

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования «Нижегородский государственный  
университет им. Н.И. Лобачевского».

603950, Нижний Новгород, пр. Гагарина, 23.

Подписано в печать \_\_.\_\_. 2015. Формат 60х84 1/16.

Бумага офсетная. Печать офсетная. Гарнитура Таймс.

Усл. печ. л. \_\_\_. Уч.-изд. л. \_\_\_.

Заказ № \_\_\_. Тираж \_\_\_ экз.

Отпечатано в типографии Нижегородского госуниверситета  
им. Н.И. Лобачевского

603600, г. Нижний Новгород, ул. Большая Покровская, 37

Лицензия ПД № \_\_-\_\_\_\_ от \_\_.\_\_.\_\_