Physical-based Animations and Mathematical Modeling

Projects

[Animations Topics]

Animations Notes:

- Put your name and title at the beginning.
- Animation duration should be at least 1 minute.
- Focus on physical phenomena, not materials models or rendering, but try to make it nice.
- Google for tutorials, self-study tutorials how to make it (it is a part of your task)
- You can use any authoring tool (eg 3dsmax, maya, blender, realflow...)
- Animations should be uploaded to youtube or any other video service and email me the link

A01: World of Domino
Place hundreds of dominos (box with texture) in some cool paths, going up/down over some obstacles, hitting balls (with different weights) that roll over and hit another balls or dominos. Some fallen dominos should slide over inclined plane, they should have different sliding friction and weight so some slide faster and longer than other. Camera should follow the falling/sliding dominos and rolling balls, making interesting details visible.

A02: A Billiard Match
Create simple billiard table with 6 holes, 2 cues, 15 numbered balls and one white ball. Your task is to animate one game, where in every round you put one ball into the hole by hitting at least 1 intermediate ball. You animate the cue striking white ball, which hits some numbered ball that hits another numbered ball which hits the hole. Balls should have different radius and weight and should physically roll and bounce. Cues should be key-frame animated.

A03: Punching Bag
Create simple human skeleton (each bone is simply a capsule or box) with a boxing glove (it should be from an elastic material). Create a punching bag (from elastic material too) hanging on a rope in front of the figure. Use Inverse Kinematics (for arm at least) to animate the arm as punch the bag. The bag should be heavy and elastic to bend when it's hit. After some strikes it should also swing on the rope. At the end the rope should tear up and the bag must fall down onto the ground. Camera should follow details

A04: North Pole
Make a simple scenery of glaciers and snowdrifts (not just a simple plane !) From the sky start to fall down big rocky 3D letters N, than O than R, T, H P, O, L, E next to each other. When the letter hit the ground it should physically collide (bound, spin...) and spray some snow particles. During the whole

animation snow particles should fall down. After all letters are settled down the snowing should increase so after a while those rocky letters start to be covered with a thin snow layer.

A05: Catching Spider
Create a skeleton of a 6 leg spider (each bone is simply a capsule or box) Make simple model of spider and apply skinning on it. Use inverse kinematics to make the spider walking in some direction. After some steps a web (make a rectangular piece of cloth with a texture of web) falls down from the sky and catches the spider.

A06: Great Fisherman
Create simple human skeleton (each bone is simply a capsule or box) with a rod (from elastic material) in the hand. Figure should be standing near some pond. Use inverse kinematics (IK) to animate how the fisherman throws the hook into the pond (hook must be on a visible nylon). When the hook hits the water simple waves should start around. After a short while the fisherman starts to twist the rope (use IK to move other hand in a rotational movement), the nylon should strain and the rod must bend. Camera can be simple static.

A07: Suit up !
Create simple human skeleton (each bone is simply a capsule or box). From cloth like material create simple t-shirt and pants around the skeleton. Make simple head (sphere is enough). Create long hair on the head (use hair or fur: google for hair in 3ds max). Use inverse kinematics for make the figure dance (some simple movements). Add wind to wave hair and cloth. Change cloth shear and bend stiffness during animation. Camera should be simple rotating around figure.

A08: World Trade Center
Create a simple scene depicting WTC towers (min two, just boxes with texture). Model simple air plane flying towards the building. Model the plane crash with the building. Plane should break apart, debris from plane and building should fall down. There should be explosion with fire and dust (you can use particle systems with textures of smoke and fire). After a while the whole building should fall down, with a huge dust and smoke (use rigid bodies as floors falling down). Camera can by static not to far from the crash.

A09: Great Fireworks
Create 3D fireworks. Design min 10 different rockets with various explosion effects. Particles should change their size, color, weight, life-time and should explode into another particles. Use nice explosion and dust particles. Change wind during animation. Camera should rotate around the fireworks.

A10: The Game of Fire
Create a rectangular piece of cloth hanging from two hooks. Next create simple model of one match and match box. Kindle the match by moving close to the matchbox. Next move it close to the piece of cloth, which starts to burn. However only letters F, I, R, E will burn, the rest of the cloth will resist (place particle emitters only in the shape of letters). Change wind during animation. Cloth and fire should move with the wind. Camera can be static.

A11: Help Firemen !
Create simple model of burning house. From broken windows and doors flames comes out (you can use fire and smoke particles). Create simple fire truck with a jet. After a while water starts to flow out of the jet, turning flames into smoke. At the end fire should be stopped and only dust comes out of the house. Camera can be static.

A12: The Beauty of Niagara
Create simple waterfall scenery. Create fluid inflow above the waterfall and simulate falling water from the rock. At the beginning show the rock without water, then start the inflow. Next create a simple boat model flowing the river above waterfall. After a while the boat should fall down with the water in the waterfall.

A13: Breaking Dam
Create simple scenery of a valley filled with water and a concrete dam. Next create a simple model of rocket (use dust and fire particles) flying towards the dam. The racket should explode after the it hits the dam. Dam breaks apart (no physical fracture is necessary, just create broken parts manually). Water should flow through the hole in the dam.

A14: Flood in the City
Create simple model of city (just boxes as buildings). Place simple cars, rubbish bins etc. onto roads. Next create large body of water flooding the city. Objects on roads should flow with the water. Camera should follow the the front wave of water.

A15: Sailing the Ocean
Create ocean-like water surface with waves. Model simple sailing ship model floating in the water. (On the water / boat contact create particle sprays). The boat should have at least one sail (use cloth material) that reacts on wind around. Change the wind power during the simulation. Create rain particles that react on wind. Camera can be static.

A16: Splash
Create splash like animation with very complex shape of splash in air. Show the shape in slow motion. See the realflow animation on youtupe for more details.

A17: Autumn Leaves
Create animation of walking man on autumn leaves on forest tract. Show the food and the leaves banging under the foot. Demonstrate the softness of leaves and softness of walking.

A18: Leaves in a Wind
Create animation autumn leaves moving in small wind in a park. Wind should take the leaves from the ground and put them in a trash. Show soft nice leave movemts so that the viewer would like to move with leaves.

A19: Walking on a Snow
Create animation of walking man on a wet snow leaving the foot tracts. Focus on very detail of snow under the food, show the detail.

A20: Melting Ice Dragon
Create animation of ice and snow dragon being melted while changing the shape.

A21: Candle
Create animation of burning candle. Candle will be lighted by matches, it will meld and deform under the temperature. Wax will fall down from the candle. Light off the candle and show the smoke.

A22: Metal Balls
Create animation of metal balls falling on metal spirals, small tracks, falling blocks and finall it andsup in aquarium.

A23: Child with Pool
Create animation of a boy playing with dirty pool on street. Show a food stepping in a pool and soft interaction with water and dirt. Show it in such a way that the viewer would like to jump in the pool.

A24: Snowing Country
Create animation of a scene during the snowing season and show the object slowly covering by a snow. Snow should create the bunch of snow.

A25: Bubbles
Create animation of small and huge bubbles. Small bubbles do not change the shape and the huge bubbles should be largely deformed. Show the elasticity of large bubbles.

A26: Fireplace
Create animation of fire being started in fireplace. Light up the fire, show growing fire, show large fire, show smaller fire, show the end of fire and the hot coal at the end.

A27: Fireplace
Create animation of fire being started in fireplace. Light up the fire, show growing fire, show large fire, show smaller fire, show the end of fire and the embers at the end.

A28: Crowd Animation
Create animation of koi fish crowd swimming in a pond. Koi is a Japanese colored carp. Use the crowd animation scripts. Make a small story with a fish. You have seen the crow animation in the Lord of the Rings.

A29: Running Cat
Create animation of running cat or dog. You should show the most realistic cat moves, tail moves and the body deforms while running. You can make it even without the motion capture data the key animation should be enough but will take time. Focus on reality.

A29: Fresh Bread
Create animation of a fresh bread being cut by a knife. Show the freshness of bread, show that it is still hot, show how it deforms before cutting, and show that it is crunchy.

[Coding Tasks]

C01: Key-framing and Parameter Interpolation

- Scene: 2d space with one box positioned at (0,0) rotated by 0 degs
- Implement parameter interpolation of position (x,y) and orientation (angle) - user can choose interpolation type
  - Nearest neighbor interpolation of a Sequence of key-frames - user can add key-frames
  - Linear interpolation of a Sequence of key-frames - user can add key-frames
  - One Cubic Bezier curve - user can move control points
- Implement time control: Change time from 0 to 1 second by a slider. Set FPS in edit-box.
  - Box should be rendered with interpolated parameters for given time

C02: Skeleton and Skinning

- Scene: 2d line-segment skeleton of figure (min 15 bones), poly-line skin around skeleton
- Implement 2d skeleton creation
  - User can create new bone by selecting parent bone (click on it) and clicking anywhere to define end of bone
  - Bones can have more child bones
  - Each bone has its position, rotation angle and length (end point can be calculated)
- Implement 2d skin creation
  - After skeleton is done, define skin around by clicking and creating poly-line
  - Generate weights automatically by calculating distance to all bones (simple all bone test). Ignore bones which are further beyond some threshold
- Implement Matrix Palette Skinning
- Implement forward kinematics
  - Select bone by clicking, change rotation angle by dragging
  - Child bones must be transformed correctly (can use transformation 3x3 matrices - they handle both rotation and translation)

C03: Inverse Kinematics

- Scene: 2d line-segment skeleton of figure (min 15 bones)
- Implement 2d skeleton creation
  - User can create new bone by selecting parent bone (click on it) and clicking anywhere to define end of bone
  - Bones can have more child bones
  - Each bone has its position, rotation angle and length (end point can be calculated)
  - User can select starting and ending bone of the IK sequence
- Implement forward kinematics
  - Select bone by clicking, change rotation angle by dragging
  - Child bones must be transformed correctly (can use transformation 3x3 matrices - they handle both rotation and translation)
- Implement inverse kinematics (relaxation by gradient calculation)
  - User can move end bone - IK solves bones in the IK sequence

C04: Particle system

- Scene: 2D fountain emitter emitting Y-Up. Gravity (9.81) Y-Down. One spherical obstacle in front of emitter
- Implement User editable emitter parameters:

◦ Emit rate: How many particles are emitted per second
◦ Emit angle: Particles are emitted in -angle +angle range around emitter direction (Y-axis) randomly
◦ Particle Life: Time in seconds each particle lives
◦ Initial velocity: How fast particles moves from emitter
• Implement position integration (ODE)
◦ Explicit Euler, Mid point and Verlet (user can choose)
• Implement simple sphere obstacle
◦ Particles resolves collisions with Newton impact law (coefficient of restitution = 0.5)
• Implement basic time control: start, pause, stop animation

C05: Uniform Grid and Pair Management
• Scene: 2d space, with n random positioned AABB boxes within some boundary, set initial velocity random
• Implement simple coherent motion
◦ Integrate positions with explicit Euler
◦ On boundary negate velocity
• Implement simple uniform grid broad phase (AddBox, RemoveBox, UpdateBox)
◦ Grid is stored in dense matrix (no spatial hashing)
• Implement simple (full matrix) pair management
• Implement hashed pair management
• Compare results (pairs) and times of both algorithms

C06: Hierarchical Grids
• Scene: 2d space, with n random positioned AABB boxes within some boundary, set initial velocity random
• Implement simple coherent motion
◦ Integrate positions with explicit Euler
◦ On boundary negate velocity
• Implement naive n^2 broad phase collision detection
• Implement hierarchical grid broad phase (AddBox, RemoveBox, UpdateBox)
◦ Grids are stored by spatial hashing in hash tables
• Compare results (pairs) and times of both algorithms
• Implement simple (full matrix) pair management

C07: Sweep And Prune
• Scene: 2d space, with n random positioned AABB boxes within some boundary, set initial velocity random
• Implement simple coherent motion
◦ Integrate potions with explicit Euler
◦ On boundary negate velocity
• Implement naive n^2 broad phase collision detection
• Implement incremental 2d SAP (AddBox, RemoveBox, UpdateBox)
• Compare results (pairs) and times of both algorithms
• Implement simple (full matrix) pair management

C08: Oriented Bounding Boxes
• Scene: Two 2D non-convex poly-line objects
• Implement object definition

- either read line segment positions from text file
- or implement user interface to create poly-line by clicking on canvas
- Implement 2D OBB near optimal fitting
- choose principal direction by rotating X-axis and Y-axis by one deg in a cycle
- for each angle (direction) project poly-line on new x,y axes and choose center as midpoint of projection, fit OBB
- finally select direction of the OOB with minimal volume (area in 2d)
- Implement binary BVT construction by Top-Down approach
- By splitting near optimal OBBs in half along principal direction (in each level should have optimal OBBS)
- Implement Tandem traversal for generated two BVTs
- Show overlapping boxes with red. Show their parent boxes up to the root with green
- When nothing is overlapping show full box hierarchy with gray (user can turn this on/off)
- Implement user interaction with object
- User can move poly-lines - OBBs are updated - rotation of poly-line is NOT required

C09: Swept Sphere Volumes (only LSS - capsules)

- Scene: Two 2D non-convex poly-line objects
- Implement object definition
- either read line segment positions from text file
- or implement user interface to create poly-line by clicking on canvas
- Implement 2D capsule near optimal fitting
- choose principal direction by rotating X-axis and Y-axis by one deg in a cycle
- for each angle (direction) project poly-line on new x,y axes and choose center as midpoint of projection, fit capsule
- finally select direction of the capsule with minimal volume (area in 2d)
- Implement binary BVT construction by Top-Down approach
- By splitting near optimal Capsules in half along direction (in each level should have optimal Capsules)
- Implement Tandem traversal for generated two BVTs
- Show overlapping capsules with red. Show their parent capsules up to the root with green
- Implement user interaction with object
- User can move poly-lines - Capsules are updated - rotation of poly-line is NOT required

C10: V-Clip: narrow phase collision detection

- Scene: Two 2D convex poly-line objects
- Implement object definition
- either read line segment positions from text file
- or implement user interface to create poly-line by clicking on canvas
- Implement simplified 2D V-Clip algorithm
- Implement method ClipVertex() for VV and VE case
- Implement method ClipEdge() for VE and EE case
- Implement user interaction with object
- User can move object (rotation is optional)
- Show closest points (features) on both geometries

C11: GJK: narrow phase collision detection

- Scene: Two 2D convex poly-line objects
- Implement object definition

- either read line segment positions from text file
- or implement user interface to create poly-line by clicking on canvas
- Implement 2D Proximity GJK algorithm
- Implement method ClosestPoint() on simplex
- Implement method SupportHC() for Hill Climbing support function
- Implement method 2D BestSimplex() for simplex refinement (simpler case as in 3D)
- Implement user interaction with object
- User can move object (rotation is optional)
- Show closest points (features) on both geometries

C12: Rigid Body Dynamics and Inertia Tensors

- Scene: 2d space with N spheres composed into one moving and rotating rigid body
- Implement object definition
- either read each sphere position and radius from text file
- or implement user interface to create spheres (position and radius) by clicking on canvas
- Implement **!!!3D!!!** rigid body dynamics (render 2D)
- Position **p(t)** and orientation **q(t)** (using quaternion)
- Linear **v(t)** and angular **ω(t)** velocity
- Linear **a(t)** and angular **α(t)** acceleration
- Scalar mass M and Inertia tensor **J**(t)
- Linear **P(t)** and angular **L(t)** momentum
- Applying force **f(t)** and torque **τ(t)** on body
- Integrate motion equation (ODE) using simple explicit Euler
- Implement Mass, Inertia tensor and Center of Mass
- Define material density as parameter
- Calculate mass and volume of each sphere
- Calculate body center of mass as weighted (mass) average of sphere centers
- Calculate Inertia tensor of each sphere (separate) and use “offset” theorem to move them into center of mass. Total inertia is then sum of all offset inertia
- Implement user interaction
- User can apply force on body by clicking and dragging. Define start of force vector (button press) and end of force vector (button release). Body should start moving and rotating w.r.t to point where to force was applied.
- During simulation show force, linear velocity and acceleration vectors.
- Restrict body position to be only inside visible area (window). Mirror linear velocity
- User can change density during simulation (mass and inertia must be recalculated)

C13: Rigid Body Dynamics and Impulse based Collisions

- Scene: 2d space with N spheres – each represent one rigid body. Draw sphere as oriented disk (draw circle and line segment from (0,0,0) to (radius,0,0) in local space, this must rotate as the body rotates)
- Implement object definition
- either read each sphere position and radius from text file
- or implement user interface to create spheres (position and radius) by clicking on canvas
- Implement **!!!3D!!!** rigid body dynamics (rendering can be only 2D)
- Position **p(t)** and orientation **q(t)** (using quaternion)
- Linear **v(t)** and angular **ω(t)** velocity
- Linear **a(t)** and angular **α(t)** acceleration
- Scalar mass M and Inertia tensor **J**(t) (mass = 1, inertia = identity, center of mass = sphere

center)
- Linear **P(t)** and angular **L(t)** momentum
- Applying force **f(t)** and torque **τ(t)** on body
- Integrate motion equation (ODE) using simple explicit Euler

• Implement impulse based collision resolution
- find colliding spheres (simple brute force n2 collision detection – no need to optimize here)
- Resolve each sphere-sphere contact by applying collision impulses

• Implement user interaction
- User can apply force on bodies by clicking and dragging. Define start of force vector (button press) and end of force vector (button release). Body should start moving and rotating w.r.t to point where to force was applied.
- During simulation show collision impulses between colliding spheres.
- Restrict body position to be only inside visible area (window). Mirror linear velocity

• Rendering – can be only 2D
- Choose plane XY (or XZ) and set position. Z=0 (or Y=0) for all objects.

C14: Mass-Spring system (infinity stiff springs)

• Scene: 2d set of particles interconnected with infinitely stiff (rigid) links (constraints).

• Implement scene definition
- either read each particle and interconnection (link) data from text file
- or implement user interface to create particles (position, mass) and links (select 2 particles to create link) by clicking on canvas
- Particles with zero inverse mass (1/mass) should be fixed in space (do not move)

• Implement position based dynamics algorithm
- Model links as simple distance constraint.
- Particles must collide with ground (bottom of the screen should be rigid ground). Collision constraint simply pushes particle above the ground.
- Gravity acts in negative y-coordinate.
- User should be able to define (change) number iterations of the constraint projection.
- No friction or restitution is needed.
- Add simple damping force to each particle ($f = - k_d v$), where k_d is damping coefficient (experiment with it) and v is current velocity of particle.

• Implement user interaction
- Users can click on particles during simulation and drag them

• Rendering – only 2D
- Draw particles as points and links as lines.

C15: Mass-Spring system (non-stiff springs)

• Scene: 2d set of particles interconnected with non-stiff springs (constraints).

• Implement scene definition
- either read each particle and spring data from text file
- or implement user interface to create particles (position, mass) and springs (select 2 particles to create spring and define some spring coefficient) by clicking on canvas
- Particles with zero inverse mass (1/mass) should be fixed in space (do not move)

• Implement position based dynamics algorithm
- Model springs as simple distance constraint with some k_s constraint coefficient ($0 < k_s < 1$)
- Particles must collide with ground (bottom of the screen should be rigid ground). Collision constraint simply pushes particle above the ground.
- Gravity acts in negative y-coordinate.

- User should be able to define (change) number iterations of the constraint projection.
- No friction or restitution is needed.
- Add simple damping force to each particle ($f = - k_d v$), where k_d is damping coefficient (experiment with it) and v is current velocity of particle.

• Implement user interaction
- Users can click on particles during simulation and drag them

• Rendering – only 2D
- Draw particles as points and springs as lines.