CA-MIRI
Elements of Animation
2ND: project

Full pipeline for character animation and crowd simulation

What's the goal?

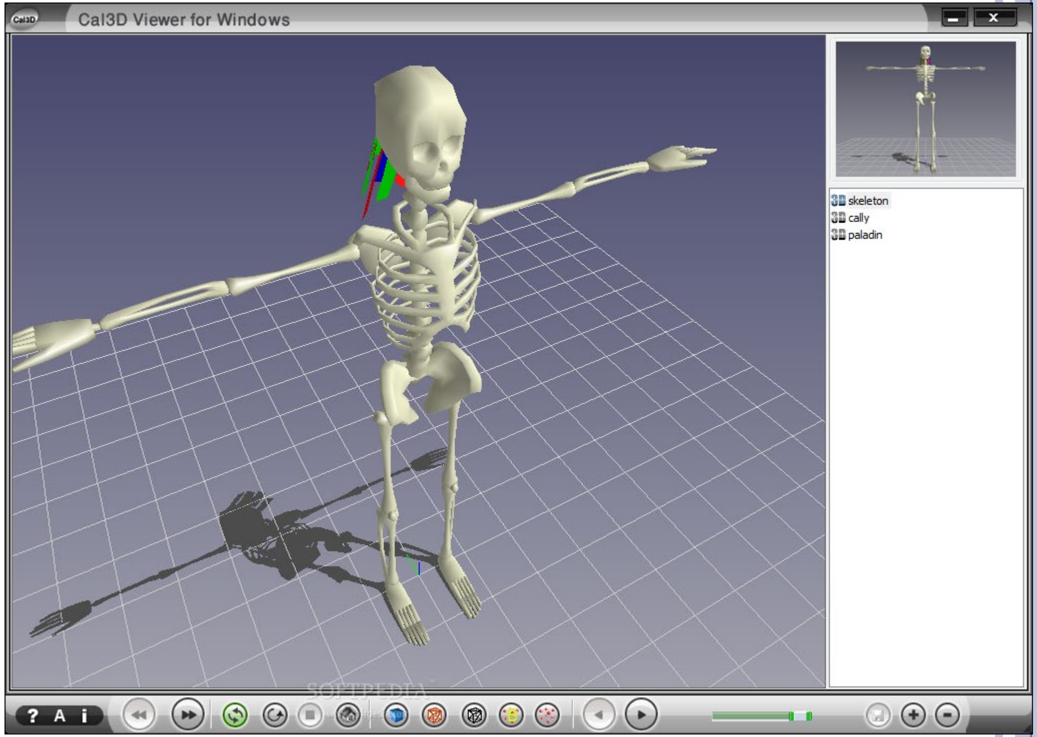
- Small crowd simulation system that integrates:
  - Character locomotion: Cal3D
  - Build a 2D grid navigation mesh
  - Path finding (A*) to move within a simple navigation mesh
  - Reynolds steering behaviors (obstacle avoidance, wander flocking, seek and flee)
  - Physically collision detection + repulsion (based on your particle system, 1st CA project)

CA-MIRI
Elements of Animation
LAB: cal3D

Nuria Pelechano Gómez

npelechano@cs.upc.edu

What is Cal3D?

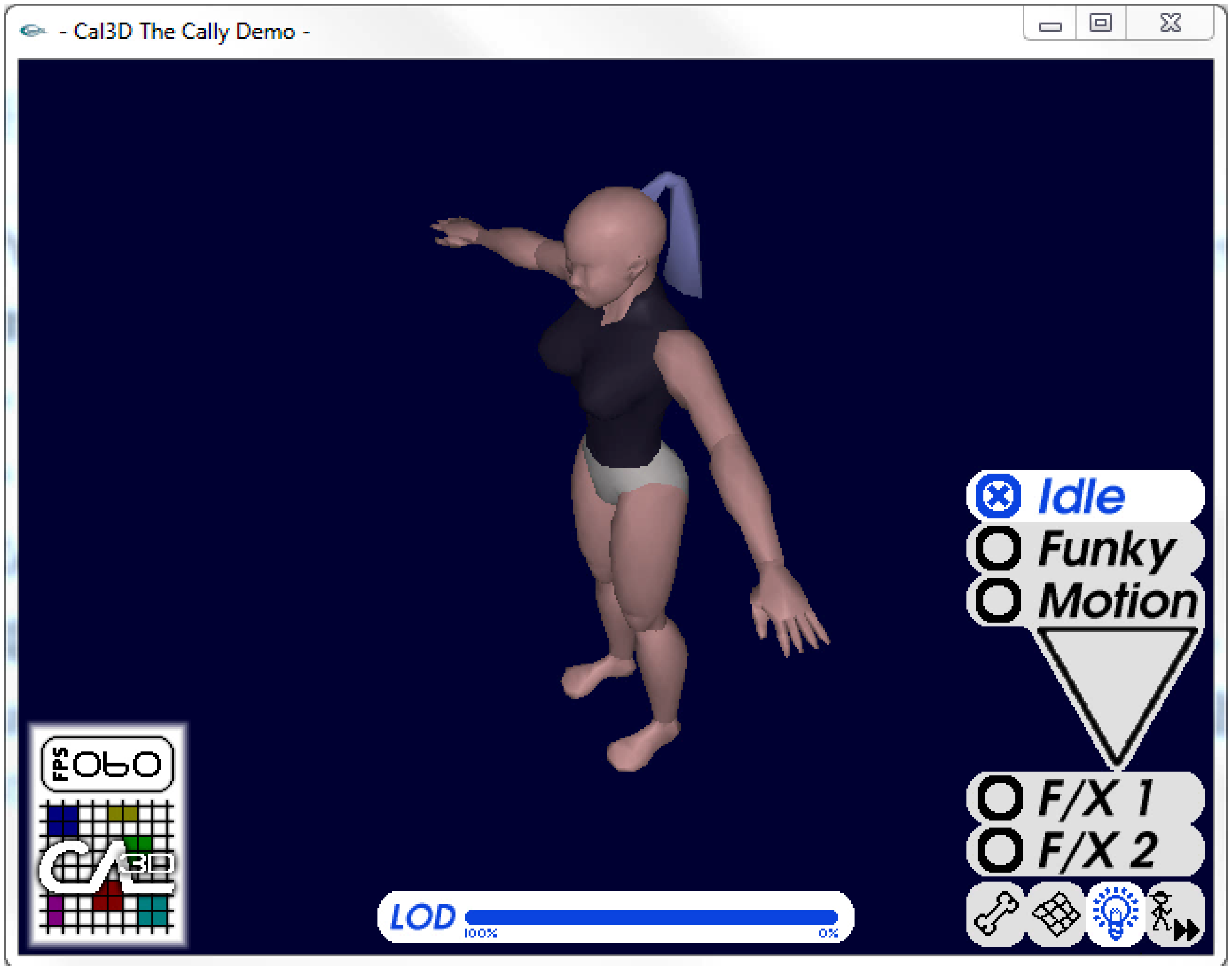


- Skeletal based 3D character animation library in C++
- Exporter (plug-ins for 3D modeling packages)
- Supports blending animations
- LOD
- include in your Visual Studio project as a dll
- Free license

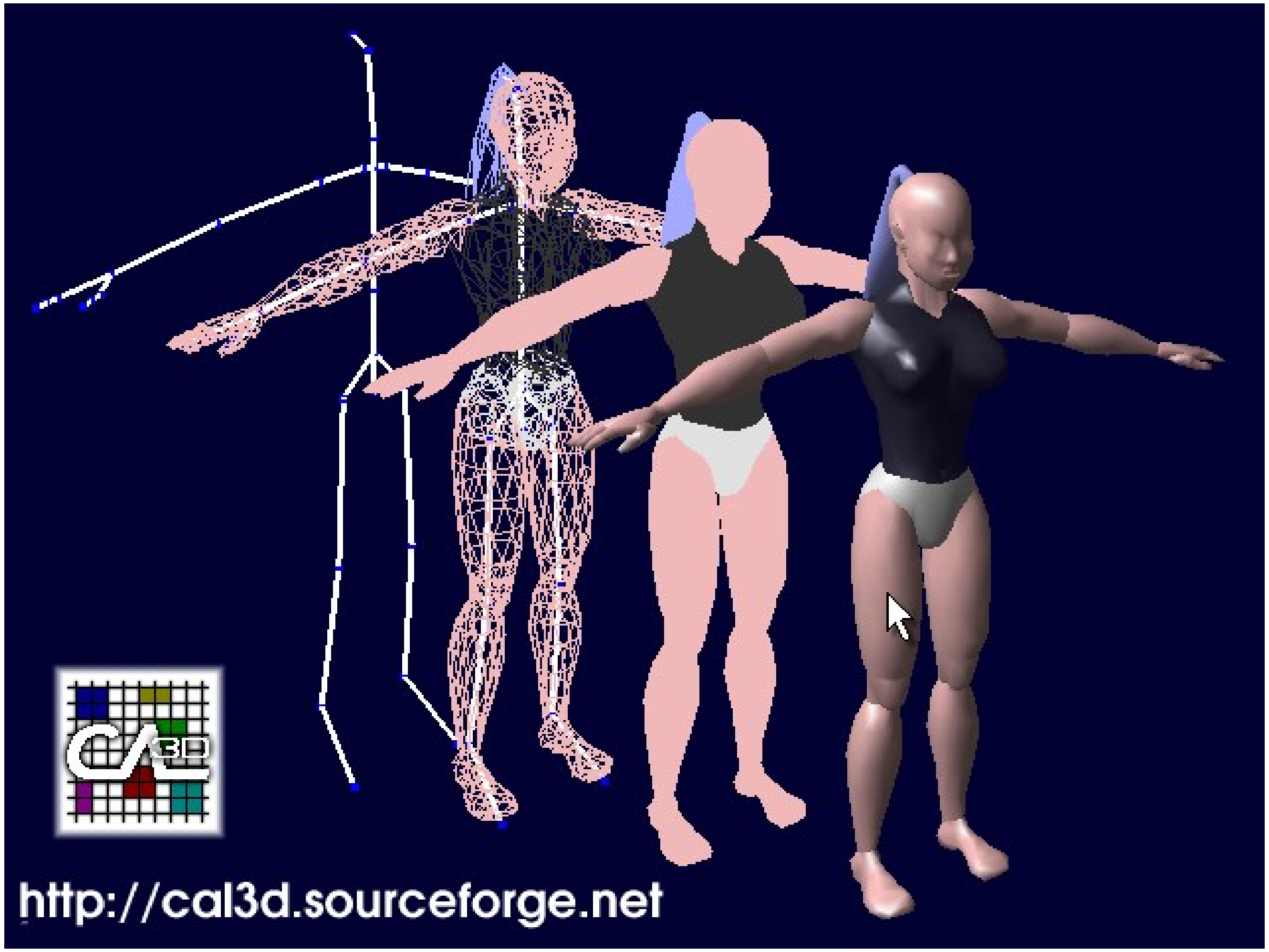
What can I do with CAL3D

- Put animated 3D characters into your interactive graphical applicatios
  - Video games
  - VR applications
- NO Graphics, but… cally demo renders using openGL
- Blending: execute multiple animations at the same time and cal3D will blend them smoothly with the given weights.
- LOD:

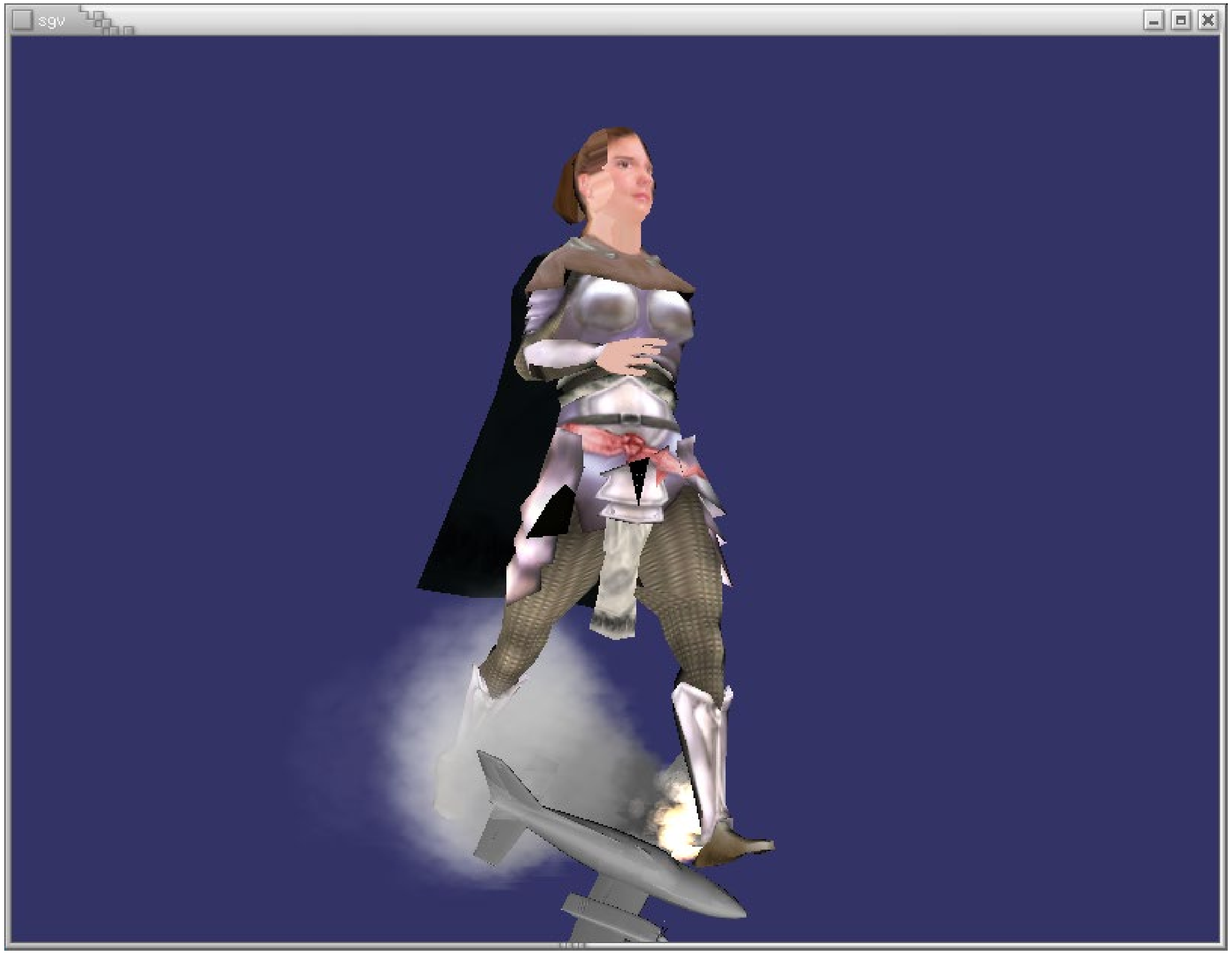
Animations and blending



Cally demo

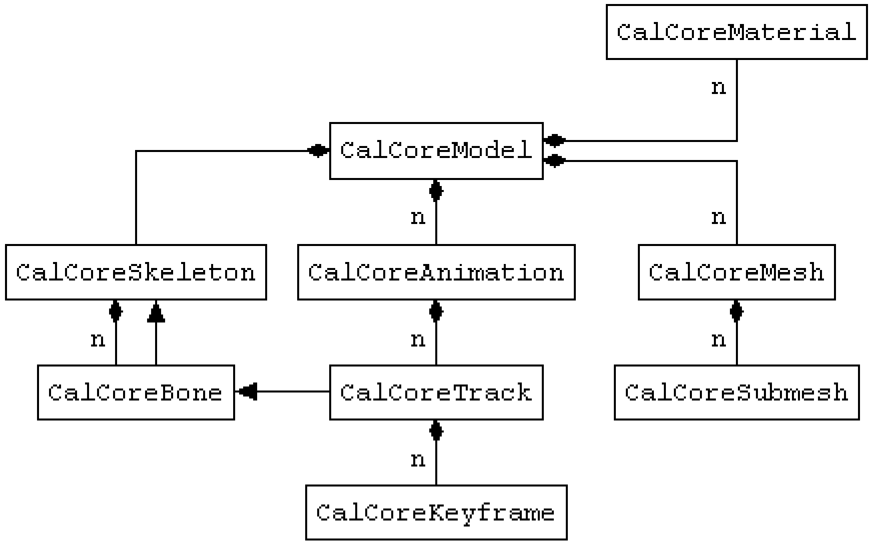


Cally demo



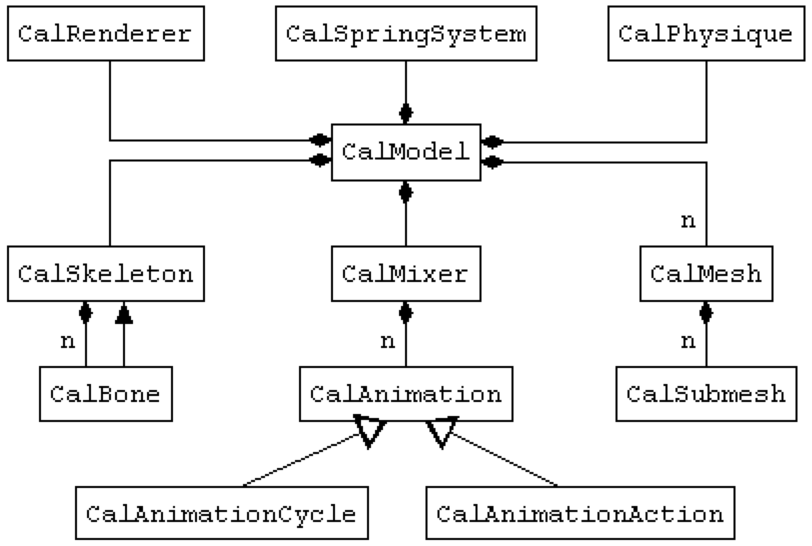
Cal3D architecture

- Core classes: contain all the shared data



Cal3D architecture

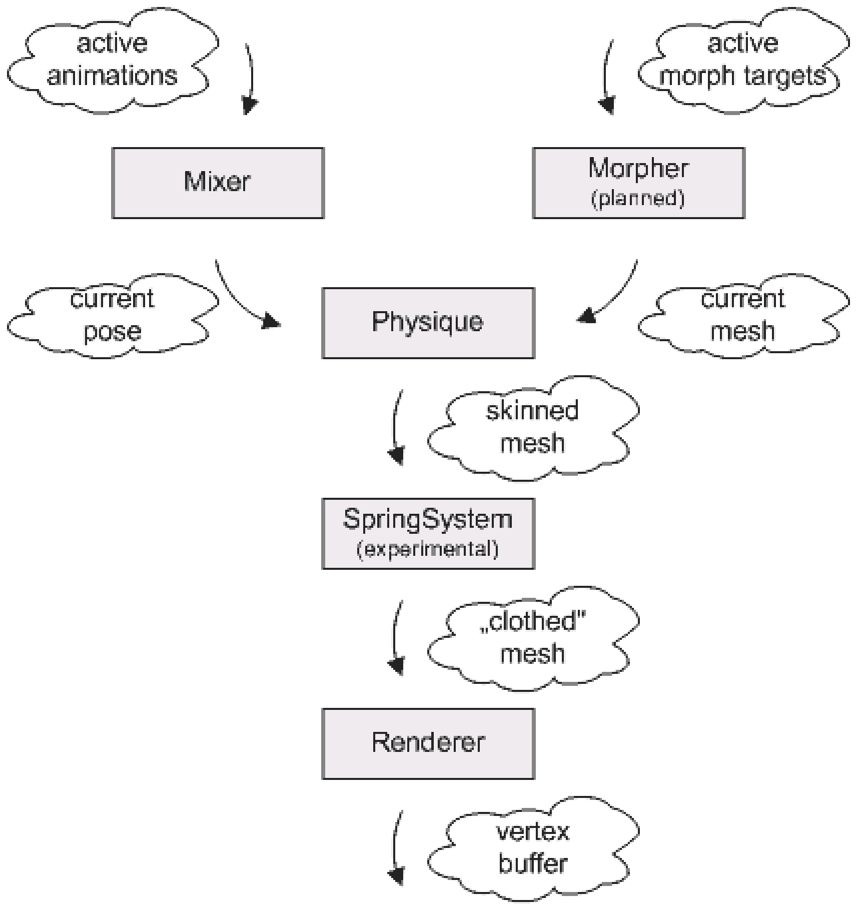
- Instance classes: contain specific data for each instance



Miscellaneous classes

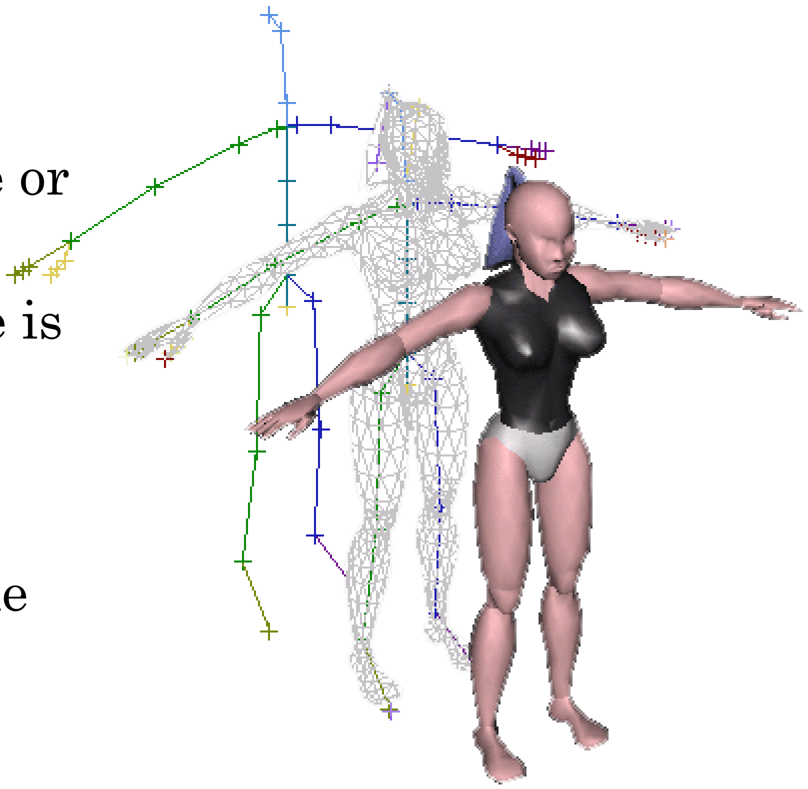
- Math components:
  - Quaternions (*CalQuaternion*)
  - Vectors (*CalVector*)
- Error handling (*CalError*)
- encapsulated platform-dependent code (*CalPlatform*)
- helper classes for loading (*CalLoader*) and saving data (*CalSaving*).

Animation pipeline



Skeletons and bones

- All mesh vertices attached to one or more bones of the skeleton
- Skinning: Once the skeleton pose is updated, the skin deforms accordingly
- Bone: defined as a relative transformation to the parent node
- Transformation = translation (vector) + rotation (quaternion)



Animations, tracks and keyframes

- Each animation stored in a core animation inside the core model. This animation contain a core track per affected bone
- The transformation for a bone is stored in several core keyframes
- For smooth transitions cal3D interpolates between keyframes
- Animations, tracks and keyframes stored per core classes
- The set of active animations and blending state store per instance

THE MIXER

- For each instance it handles
  - Addition/removal of animations from the active animation set
  - Update of active animation states (fade in/out)
  - Weighted blending and prioritized overlay
  - Update of skeleton pose
- Animations: cycle or action

Meshes and submeshes

- Models can have one or more meshes
- Each mesh consists of triangular faces
- Each submesh contains a list of vertices and a list of faces
- Each vertex contains information about the influenzes from the assigned bones
- Core model holds all meshes for a model type
- Model instances: material and LOD

The renderer

- For each submesh you execute a number of steps:
  1. Set the current mesh/submesh for the data query.
  2. Get the material data.
  3. Get the deformed vertex data.
  4. Get the transformed normal data.
  5. Get the mapping coordinate data.
  6. Get the face data.
  7. Set the rendering state in the used graphic API, and render the data.
- Most of the data is returned in a user-allocated array to make usage of vertex- buffers as easy as possible.

USAGE

- Include cal3d.h
- Your project has to be linked to the cal3d library code
  - Download from raco the compiled version
  - Or open cal3d.dsw workspace and do “rebuild all”

Core model handling

- Create an instance of the core model

```
myCoreModel = new CalCoreModel(“dummy”)
```

Core model handling

- Load skeleton

```
myCoreModel->loadCoreSkeleton("hero.csf")
```

- Load animations

```
int idleAnimationId, walkAnimationId;

idleAnimationId =
myCoreModel->loadCoreAnimation("hero_idle.caf");
walkAnimationId =
myCoreModel->loadCoreAnimation("hero_walk.caf");
…
```

CORE MODEL HANDLING

- Load mesh

```
int upperBodyMeshId, lowerBodyMeshId;

upperBodyMeshId =
myCoreModel->loadCoreMesh("hero_upperbody.cmf");
lowerBodyMeshId =
myCoreModel->loadCoreMesh("hero_lowerbody.cmf");
```

- Load materials

```
int upperBodyChainmailMaterialId, upperBodyPlatemailMaterialId;

upperBodyChainmailMaterialId =
myCoreModel->loadCoreMaterial("hero_upperbody_chainmail.crf");
upperBodyPlatemailMaterialId =
myCoreModel->loadCoreMaterial("hero_upperbody_platemail.crf");
```

Model instance handling

- After the initialization of a core model, you can create as many model instance as you wish
- core objects (skeleton, meshes, materials, bones, etc.) contain the data that will not change at runtime and can therefore be shared between multiple independent characters.
- non-core classes contain data about things that frequently change, such as the animation state and mesh data. This allows the instances to share data and still be able to be animated independent of one another.

Model instance handling

- Creation

```
CalModel *myModel;

myModel = new CalModel(myCoreModel);
```

- Attachment and detachment of meshes

```
if(!myModel.attachMesh(helmetMeshId))
  {    // error handling ...
  }

if(!myModel.detachMesh(helmetMeshId))
  {    // error handling ...
  }
```

Model instance handling

- LOD

```
myModel.setLodLevel(0.5f);
```

- Materials

```
myModel.setMaterialSet(CHAINMAIL_MATERIAL_SET);

myModel.getMesh(upperBodyMeshId)-
>setMaterialSet(PLATEMAIL_MATERIAL_SET);
```

Model instance handling

- Types of animations:
  - Cycles
  - Actions (one-time)

Model instance handling

- Functions to control cycles
  - blendCycle() adjusts the weight of a cyclic anim in a given amount of time
  - clearCycle() fades out an active cycle in a given amount of time

```
myModel.getMixer()->clearCycle(idleAnimationId, 0.3f);
myModel.getMixer()->blendCycle(walkAnimationId, 0.8f, 0.3f);
myModel.getMixer()->blendCycle(limpAnimationId, 0.2f, 0.3f);
```

- Functions to control actions
  - executeAction() execute an action animation, with fade in and fade out delays

```
myModel.getMixer()->executeAction(waveAnimationId, 0.3f, 0.3f);
```

Model instance handling

- State update:
  - Smooth motion of the models
  - Involves: evaluating new time and blending values for the active animations and calculate resulting pose
  - Argument: elapsed seconds

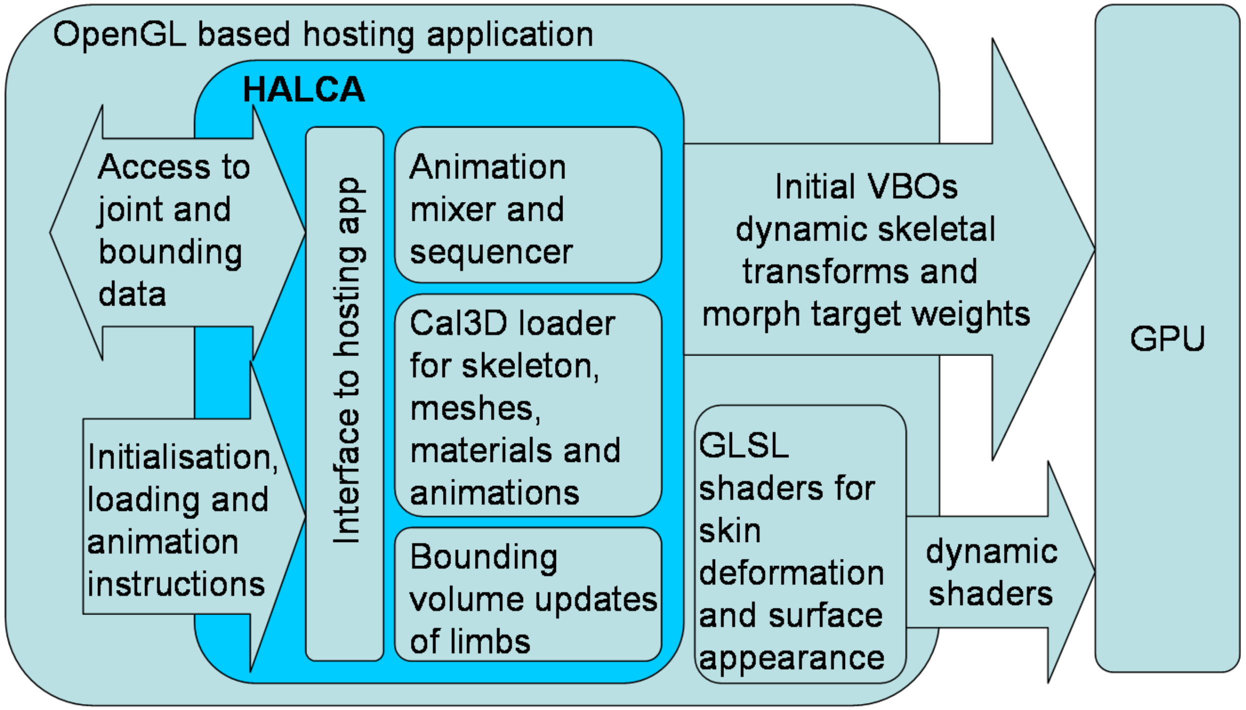
```
myModel.update(elapsedSeconds);
```

- Time warping!!!

Is there anything better out there?

- Yes: HALCA

http://www.lsi.upc.edu/~bspanlang/animation/avatarslib/doc/


OpenGL based hosting application
HALCA
Access to joint and bounding data
Initialisation, loading and animation instructions
Interface to hosting app
Animation mixer and sequencer
Cal3D loader for skeleton, meshes, materials and animations
Bounding volume updates of limbs
Initial VBOs dynamic skeletal transforms and morph target weights
GLSL shaders for skin deformation and surface appearance
dynamic shaders
GPU

1ST LAB: CAL3D

Integrate cal3D in your particle simulation system

Where do I start?

- Clean-up your particle simulation system to get a 2D version of your code (Y=0)
- Get each particle (start with 1-5) and assign a random velocity vector in XZ
- Limit your “world” to a 15x15 square
- When particle reaches the limit of your word “bounce back”
- For each particle create a cal3D model

- Add a Cal3D model instance for each particle (character heights ranging from 1.6 to 1.85).
- Move agents in straight lines with different velocities until reaching the limits of the terrain
- Bounce their velocity vector
- Perform collision and repulsion between particles
- Turn the terrain into a 20x20 square, of cells 1m^2
- Each cell with be a node of your future navigation graph for path finding (start thinking on your future data structure)