FEMFX Overview & Integration

Advanced Micro Devices, Inc.

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD is a trademark of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Introduction

FEMFX is a multithreaded CPU library for deformable material physics, using the Finite Element Method (FEM). Solid objects are represented as a mesh of tetrahedral elements, and each element has material parameters that control stiffness, how volume changes with deformation, and stress limits where fracture or plastic (permanent) deformation occur. The model supports a wide range of materials and interactions between materials. We intend for these features to complement rather than replace traditional rigid body physics. The system is designed with the following considerations:

- Fidelity: realistic-looking wood, metal, plastic, even glass, because they bend and break according to stress as real materials do.
- Deformation effects: non-rigid use cases such as soft-body objects, bending or warping objects. It is not just a visual effect, but materials will resist or push back on other objects.
- Changing material on the fly: you can change the settings to make the same object behave very differently, e.g., turn gelatinous or melt
- Interesting physics interactions for gameplay or puzzles

The library uses extensive multithreading to utilize multicore CPUs and benefit from the trend of increasing CPU core counts.

Features

- Elastic and plastic deformation
- Implicit integration for stability with stiff materials
- Kinematic control of mesh vertices
- Fracture between tetrahedral faces
- Non-fracturing faces to control shape of cracks and pieces
- Continuous collision detection (CCD) for fast-moving objects
- Constraints for contact resolution and to link objects together
- Constraints to limit deformation
- Dynamic control of tetrahedron material parameters
- Support for deforming a render mesh using the tetrahedral mesh

Files

1. Documentation
    - amd_femfx\docs\
2. Library
    - amd_femfx\inc\ : Public API (Look first at AMD_FEMFX.h)
    - amd_femfx\src\ : Implementation files
3. Sample code
    - samples\FEMFXViewer\: Basic sample; displays tet meshes and debug information
    - samples\common\TestScenes.*: Setup for tech demo scenes used by FEMFXViewer
    - samples\common\FemResource.* : Container for the data loaded from a .FEM file

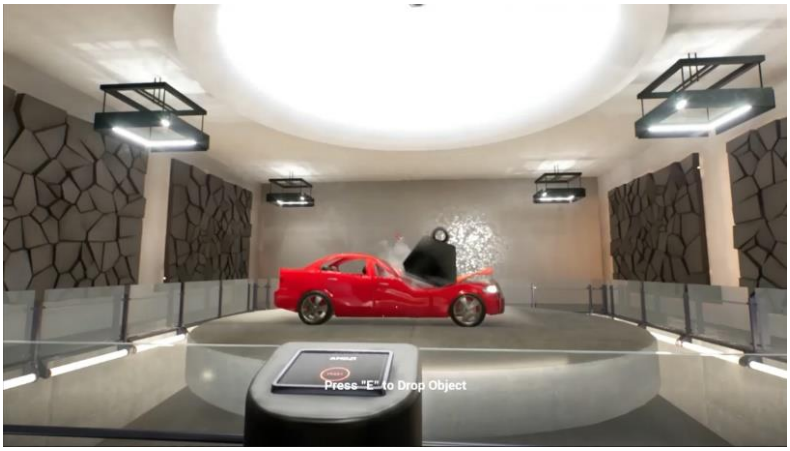
- samples\common\LoadFem.*: Code for parsing a .FEM and loading into an FEMResource
- samples\common\RenderTetAssignment.* : Assigns rendering mesh vertices to tetrahedra for skinning
- samples\ExampleRigidBodies* : Basic demonstration of interfacing an external rigid body system with FEMFX library
- samples\sample_task_system* : Sample task system library interfaced with FEMFX

4. Houdini plugin for content creation
    - houdini16.5\AMD_FEM_Assets.otl
5. External dependencies
    - external\glfw\ : Path to install GLFW for viewer; can download Windows pre-compiled binaries from https://www.glfw.org/download.html
    - external\json-develop\: Path to install json parser used in LoadFemFile.*; can download from https://github.com/nlohmann/json
6. Some individual components
    - amd_femfx\inc\Vectormath* : 3D vector/matrix math
    - amd_femfx\src\PrimitiveCollision* : Triangle and box collision operations, intersection and CCD
    - amd_femfx\src\Common\FEMFXSvd3x3.* : 3x3 SVD
    - amd_femfx\src\Threading* : Async parallel-for and task graph support
    - samples\sample_task_system\TL* : Async task system implementation

Demo Use Cases

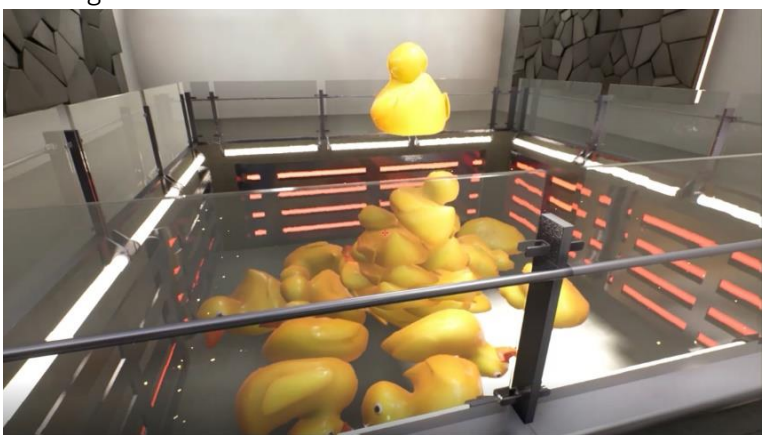
Bending/breaking wood

Denting metal



Elastic deformation

Melting



Use-case Recommendations

Our system is designed for physics objects with typically fewer than a thousand tetrahedra per object. Performance may be helped by reducing the resolution of the tetrahedral mesh, and often a coarser mesh still provides enough degrees of freedom for interesting deformations. For example, the duck tetrahedral model above is very simple with only 33 tetrahedra, but the contact and deformations still show realistic subtleties.

A general recommendation is to use this with materials that have noticeable deformation. In some cases this deformation can be slight but still add fidelity or variation to the way game objects behave.

We recommend uses that allow the player to interact with the physics and have control over the specific deformation and fracture response. This can add to the gameplay experience rather than being just a visual effect.

Integration Overview

The FEM physics simulation is performed on the CPU by the FEMFX library, which provides an API for creating and managing mesh objects and scenes. This library requires some external functions to be provided by the application through callbacks, for example, to implement threading operations. Additional information is provided in the main API header amd_femfx\inc\AMD_FEMFX.h

Content may be created using our custom Houdini tools, which outputs to a .FEM file format. This needs to be imported and converted to an application-specific format. We provide some sample files which parse and load data from the .FEM file (samples\common\LoadFemFile.[h,cpp], samples\common\FemResource.[h,cpp])

FEM rendering is expected to be customized by the application but is assisted by library API, and by data included in the .FEM file. An example rendering approach which resembles vertex skinning is demonstrated in our Unreal Engine plugin, available for UE developers here: https://github.com/GPUOpenSoftware/UnrealEngine/tree/FEMFX-4.18. This uses a vertex shader to compute render vertex positions and normals using weighted sums of tetrahedra positions and rotations. Further information on the Unreal plugin is available here:

https://www.youtube.com/watch?v=IYClvszCCPA. We also provide generic sample code outside the plugin to support this skinning method (in samples\common\RenderTetAssignment.[h,cpp]).

Integrating FEMFX with existing rigid body physics is supported by API and sample code but may still take significant effort if two-way interaction is required.

Main Challenges

Rigid-body Interfacing

We believe it is best to combine FEM physics with rigid-body physics. Rigid body physics can be faster per object, and a good approximation for some materials. Ideally FEM could be used selectively for content where it has the greatest benefit and be a more incremental addition to existing game physics systems.

However, two-way interaction between these objects is non-trivial. We have paved the way for this by use of an iterative constraint solver in the FEM simulation which is compatible with solvers commonly used in rigid body simulation. The FEMFX API has some interfaces to support a coordinated constraint solve, and we include a sample for this. However, rigid body engines typically do not have API for this, and the two-way interfacing may require some non-trivial changes to the rigid body engine code. Alternatively, the library includes some internal support for rigid body physics, however this is still very limited.

In contrast one-way interaction is much simpler, for example, if FEM is only affected by moving some its vertices kinematically, or by collisions with kinematic objects. In this case the FEM system and rigid body system can stay independent.

FEM content authoring

The material parameters provide a lot of flexibility but there is a learning curve for understanding these and achieving desired material behaviors. More explanation of the material parameters is provided in the next section.

The user must also be careful of the “conditioning” of the meshes, which can affect the performance and robustness of the simulation. This may require changes to material parameters and avoiding modeling tetrahedra that are too skinny or flat. The library includes a function to estimate conditioning to provide feedback.

Performance tuning

Performance is also very sensitive to scene complexity and content. This may be improved by reducing vertex/tetrahedron counts in meshes, and limiting amount of contact between objects, particularly the size of groups of contacting objects. It may be helpful to disable unnecessary contact using the provided collision group interface. In addition, it may be helpful to disable some objects when no longer needed, using the provided FmEnableSimulation() function.

Material Parameters

The material parameters of this physics model allow for a wide range of behaviors for individual and interacting objects. Note these values may not strictly follow realistic values and may need to be adapted for each object to achieve the desired look, since behavior may change with mesh geometry and resolution:

- Young's Modulus: Greater values increase stiffness of the material. Values can range widely, for example from 10^3 to 10^7 or higher for some of the more rigid objects we have created. Values may need to be adapted with changes to mesh geometry or masses.
- Poisson's Ratio: Determines how the material preserves volume and bulges when compressed, with 0 causing none. The maximum is 0.5 but we tend to use lower values like 0.3 to improve the conditioning mentioned above.
- Rest Density: Density of the tetrahedron at rest which will determine mass, based on the rest volume. It is also possible to override the total mass of an object with FmSetTotalMass()
- Fracture Stress Threshold: Amount of stress needed for fracture to occur. Stress is caused by deformation, but also affected by Young's and Poisson's values, so this value may need to be adjusted with changes to those.
- Plastic Yield Threshold: Amount of stress needed for the mesh to start plastically deforming, i.e., holding the deformed shape. Basically, this value gives a way to tune how much deformation is needed before it starts to become permanent.
- Plastic Creep: Value >= 0 and <=1. This is the portion of elastic deformation converted to plastic deformation, so lets you tune how quickly the plastic deformation sets in.
- Plastic Min and Max: The min value is > 0 and <= 1, and max value is >= 1. This limits the amount of permanent compression or stretching that can be caused by plastic deformation, which prevents some conditioning issues (example default values may be min of 0.25 and max of 4.0)
- Lower and Upper Deformation Limits: Values are unlimited if 0. Otherwise the lower value should be > 0 and <= 1, and the upper value should be >= 1. These values set limits on compression or stretching of a tetrahedron. When values are non-zero and deformation is found outside a limit, a constraint is created to correct the deformation. Note these limits add CPU cost and can result in jitter, so it may be preferable to increase Young's Modulus to limit deformation. However, these can be useful, for example, to prevent too much compression of softer objects.

Tuning for conditioning

Masses, Young's Modulus, Poisson's Ratio, and mesh geometry all contribute to numerical conditioning, and we put in the FmCheckTetMeshCondition() function to give some feedback for improving this. The current rough recommendation is to keep the value under 20000 (given current defaults for iteration counts). Increasing mass, lowering Young's or Poisson's can reduce this. Models with skinny or flat tetrahedra can worsen this and limit the flexibility in setting material parameters. Note that with the Houdini authoring tools, one creates a tetrahedral mesh by first modeling a surface mesh and using an automatic tetrahedralizer, so a first requirement is to avoid skinny triangles when modeling the surface mesh. The Houdini tools include a feature to highlight high-aspect-ratio tetrahedra.

Kinematic Flags

The library is designed to support setting some mesh vertices as kinematic (using the flag FM_VERT_FLAG_KINEMATIC), making them unaffected by the simulation and free for the application to move by directly setting velocity. Note, the simulation step will update positions of these vertices according to the velocities input, however, the application can set position directly to prevent drift. One use of these is to anchor FEM objects to the environment, and the default velocity is zero.

Note if all vertices of a mesh are kinematic, for instance a static object in the environment, one should indicate the object is kinematic in the setup parameters, which can reduce some costs in the simulation step.

Some care is needed when objects containing kinematic vertices can be fractured. An artifact that can occur near the kinematic vertices is that a piece may break off which contains only one or two kinematic vertices, which can cause it to swivel on a ball or hinge joint. We provide the option to automatically remove the kinematic property on such vertices, if the user sets a flag (FM_VERT_FLAG_KINEMATIC_REMOVABLE). Also, for any vertices with this flag, the simulation will automatically remove this property when stress on the vertex is detected to be greater than a specified threshold. This gives the appearance of fracture between the object and the environment.

Alternatively, one can use a portion of an object's tetrahedra as an anchor to the environment, which can give more flexibility to design the shape of the fracture. In this case one should also set these tetrahedra as kinematic (using FM_TET_FLAG_KINEMATIC), as well as setting their vertices kinematic.

Note that if any tetrahedra of a mesh are anchored inside another kinematic object or outside the collision box for the scene, all vertices of these tetrahedra should be set as kinematic. Otherwise, collision response will attempt to correct the overlap causing jitter and deformation artifacts.

Sleeping

The library has a feature to put a slow-moving meshes to sleep, which freezes the movement and eliminates most CPU cost associated with the mesh. Groups of contacting meshes are put to sleep at once, and all bodies in this group are required to fall below thresholds for speed of movement.

Note that this may not work well with poorly conditioned meshes, or due to solving errors with larger piles of contacting objects, or when objects are positioned such that constraint fighting takes place as described above. Alternatively, the method may freeze some objects in an imbalanced position.

One can tune the thresholds for each object using the calls FmSetSleepMaxSpeedThreshold(), FmSetSleepAvgSpeedThreshold(), and FmSetSleepStableCount() functions.

Constraints

A scene can include constraints between FEM objects, which can be authored using the Houdini tools. These include:

- Glue constraints: These will pin two objects together at specified object locations. These can be used to create compound objects. They include a force threshold to be automatically broken (disabled). Also, one can require a minimum number of glue constraints between two objects, below which the glue constraint will be broken. For example, this can be used to prevent one or two glue constraints behaving like a ball or hinge joint.
- Plane constraints: These consist of 1-3 plane constraints, which may be one-sided (e.g., repulsion forces only). These can be used for example to force space between objects for greater robustness against parts of objects getting entangled. The plane normals must be updated each frame by the application.
- Angle constraints: Currently these constraints only support creating a hinge joint between rigid bodies.

Note that the constraint solver method used in FEMFX may sometimes converge too slowly and constraints may not be satisfied well with the current number of maximum iterations. This can for example result in gaps between glued points. Some advice for constraints:

1. Try creating a single FEM mesh when possible rather than gluing separate meshes. Use of material settings and non-fracture regions can achieve the effect of separate meshes.
2. Use kinematic vertices instead of glue to pin objects to the environment.
3. Reduce number of constraints when possible.
4. If collisions are not disabled between objects, make sure the glue does not force them to overlap (be closer than the scene's contact gap), or else the collisions and glue will fight each other and result in glitches.
5. Keep glue positions close to a tetrahedral mesh.

Rendering

Rendering must be handled on the application side, but the FEMFX library can provide API to support it and we have demonstrated example rendering techniques in the UE plugin.

Deformation

To render FEM objects in the UE plugin, we create a separate render mesh and deform it using the FEM simulation mesh. This can be an effective way to improve the apparent simulation detail using only visual detail.

Deforming the render mesh can be accomplished by assigning each render vertex to a tetrahedron in the tetrahedral mesh and computing the render position using a barycentric weighted sum of the tetrahedron vertices. The assignment and barycentric weights can be precomputed during preprocessing of the object. We provide sample code to perform this preprocessing in in samples\common\RenderTetAssignment.[h,cpp]. At run time the weighted sum can be performed in a vertex shader.

Fracture

Fracture in the simulation mesh occurs between tetrahedral faces and can also be limited by creation of non-fracture regions. We support fracture in the render mesh by authoring the render mesh in pieces corresponding to possible fractures in the simulation mesh. Before fracturing occurs, the vertices of these pieces are assigned to tetrahedra as before. However, when fractures occur, we may re-assign vertices so all vertices of a rendering piece are assigned to the same fractured piece of the simulation mesh.

We precompute the possible tetrahedron assignments for each render vertex and at run time use an offset value to select between different assignments. In addition, render mesh vertices with the same set of assignments will share an offset value, and can reduce the cost of changing the assignments for these vertices.