Apex destructible

Description

Apex is add-in for PhysX library, that allows creating and rendering realtime, physically correct object destructions (it has several more applications, but this paper covers only destructible usage). Basic usage pipeline consist of creating special destructed object from ordinary solid model using 3D Studio max plugin and creating special type of game object in Level Editor based on data exported in previous state. Apex game object can be in 2 states: solid state (this state is used to configure position and orientation of object in game level), and destructible state (this state enables Apex physics simulation for object, and used for gameplay destructions).

To be operable in game environment Apex SDK requires implementation of several classes, and following text describes all important aspects of them.

For detailed description please consult Apex SDK documentation.

Associated classes and structures

ApexWorld	Apex scene wrapper
ApexActorBase	Abstract class represents access interface to the apex actor object used for destruction visualization
ApexActor	Actual actor with enabled physics simulation
ApexPerviewActor	Stub class used for apex object configuration and placement
UserRenderer	Implementation of apex renderer class that used for rendering apex objects
UserRenderResourceManager	Implementation of apex resource manager interface, used for renderer dependent resources creation.
UserRenderVertexBuffer	Implementation of apex vertex buffer interface, that represents engine dependent vertex buffer
UserRenderIndexBuffer	Implementation of apex index buffer interface, that represents engine dependent index buffer
UserRenderBoneBuffer	Implementation of apex bone buffer interface, that represents engine dependent bone buffer
UserRenderResource	Implementation of apex render resource interface, that contains all necessary pointers to data buffers needed for 3d object rendering
ApexFileStream	Implementation of file stream interface, that used to read engine dependent file data.
obj_ApexDestructible	Game object that represents apex destructible entity.

class ApexWorld

Apex scene is designed as wrapper above PhysX scene, and this class used to wrap PhysX scene simulation calls. It is used for several utility operations too.

Sources: ApexWorld.h, ApexWorld.cpp

Important methods

void Init()
Initialization of Apex SDK, creation of apex scene and linking it with PhysX scene.

void StartSimulation()
Start physics simulation. Apex simulation loop should replace original PhysX simulation function calls.

void ApplyAreaDamage(float damage, float momentum, const r3dPoint3D& position, float radius, bool falloff)
Apply area damage to all apex objects.
Parameters:
 damage – amount of damage at epicenter
 momentum – magnitude of damage impulse
 position – damage epicenter location
 radius – maximum influence distance
 falloff – enable/disable linear falloff from epicenter to edge (defined by radius)
See NxDestructibleActor::applyRadiusDamage for detailed description

void ApplyDamage(float damage, float momentum, const r3dPoint3D &pos, const r3dPoint3D &direction)
Apply damage at point
Parameters:
 damage – amount of damage at damage point
 momentum – magnitude of damage impulse
 pos – position of damage force
 direction – direction of damage force

void ConvertAllToDestructibles()
Convert all apex object in level to "destruction enabled" state. In this state apex object can be affected by external forces and destroyed.

void ConvertAllToPreview()
Convert all objects to preview state. In this state apex object can be moved by level editor controls, and does not affected by physics forces. This state is used for object setup and configuration.

class ApexActorBase

This class is a wrapper of apex object and control it behavior. Class is abstract, and cannot be directly instantiated. There are to derived classes: ApexActor – actual apex object, with enabled physics simulation, and ApexPreviewActor – stub class with disabled physics simulation. Latter class is used for apex object setup and configuration.

Sources: ApexActor.h, ApexActor.cpp

Important methods

void Update()

Updates simulation state of apex object. For ApexPreviewActor do nothing.

void Draw()

Render apex object.

void SetPosition(const r3dVector &pos)

Do nothing for actual apex object (with enabled physics simulation state apex object movement is disabled). Move object to specified position in case of preview object.

void SetRotation(const r3dVector &angles)

Do nothing for actual apex object. Rotate object to specified angles in case of preview object.

ApexActorTypes Type()

Get type of object: actor or preview. ApexActorTypes have following definition:

```
enum ApexActorTypes
{

        TypeApexDestructible = 0,
        TypeApexPreview
};
```

ApexActorBase * ConvertToOpposite()

Convert object to opposite type. I.e. actor->preview, or preview->actor.

void ApplyAreaDamage(float damage, float momentum, const r3dPoint3D& position, float radius, bool falloff)
Apply area damage to actor. In case of preview object do nothing. See ApexWorld::ApplyAreaDamage

void ApplyDamage(float damage, float momentum, const r3dPoint3D &pos, const r3dPoint3D &direction)

Apply point damage to actor. In case of preview object do nothing. See ApexWorld::ApplyDamage

class UserRenderer

To render apex objects Apex SDK requires implementation of user render class that will draw apex provided vertex and index buffers.

Sources: ApexRenderer.h, ApexRenderer.cpp

Important methods

void renderResource(const NxApexRenderContext& context)

Render given resource. This function is called by Apex SDK. context parameter contains pointers to all necessary index, vertex buffers as well as materials that needed to draw 3d model.

bool CreateFullVertexDescription(const NxApexRenderContext& context, D3DVERTEXELEMENT9 *ve, size_t count) const

Create array of D3DVERTEXELEMENT9 type from apex specific context structure, with valid vertex description data. Created structure is hashed by HashVertexDescription function, to prevent duplicate creations.

uint32_t HashVertexDescription(D3DVERTEXELEMENT9 *ve, size_t count) const

Create a hash for d3d specific vertex description array.

class UserRenderResourceManager

Apex SDK resource creation requests are handled by this class.

Sources: ApexRenderer.h, ApexRenderer.cpp

Important methods

NxUserRenderVertexBuffer * createVertexBuffer(const NxUserRenderVertexBufferDesc& desc)

Create vertex buffer from apex provided description. There is associated release method.

NxUserRenderIndexBuffer * createIndexBuffer(const NxUserRenderIndexBufferDesc& desc)

Create index buffer from apex provided description. There is associated release method.

NxUserRenderBoneBuffer * createBoneBuffer(const NxUserRenderBoneBufferDesc& desc)

Create bone buffer from apex provided description. Bone buffer defined as array of matrices. It is used for object debris position and rotation specification (similar to skinning). There is associated release method.

NxUserRenderResource * createResource(const NxUserRenderResourceDesc& desc)

Create user render resource from apex provided description. User render resource is structure with vertex, index, bine buffer pointers with appropriate offsets. This data unambiguously defines 3d mesh that is used for rendering.

physx::PxU32 getMaxBonesForMaterial(void* material)
Return maximum number of bones that can be used simultaneously during rendering. In case of software (CPU) skinning method, this number can be infinitely big. In case of GPU skinning, we are limited by number of vertex shader registers.

class UserRenderVertexBuffer

User implemented vertex buffer class, that derived from NxUserRenderVertexBuffer interface.

Sources: ApexRenderer.cpp

Important methods

void writeBuffer(const NxApexRenderVertexBufferData& data, physx::PxU32 firstVertex, physx::PxU32 numVertices)

Write vertex data provided by apex into d3d vertex buffers.

Parameters:
data – vertex buffer data provided by apex SDK.
firstVertex – index of first vertex to read.
numVertices – number of vertices to read.

class UserRenderIndexBuffer

User implemented index buffer class, that derived from NxUserRenderIndexBuffer interface.

Sources: ApexRenderer.cpp

Important methods

void writeBuffer(const void* srcData, physx::PxU32 srcStride, physx::PxU32 firstDestElement, physx::PxU32 numElements)

Write indices provided by apex into d3d index buffers.

Parameters:
srcData – source data to read from.
firstDestElement **–** first index to read
numElements – number of indices to read.

class UserRenderBoneBuffer

User implemented bone buffer class, that derived from NxUserRenderBoneBuffer

Sources: ApexRenderer.cpp

Important methods

void writeBuffer(const NxApexRenderBoneBufferData& data, physx::PxU32 firstBone, physx::PxU32 numBones)

Write bone matrices provided by apex into internal memory buffer.

Parameters:
data – bone buffer data provided by apex SDK.

firstBone – index of first bone to read.
numBones – number of bones to read.

class UserRenderResource

User implemented class that encapsulates all necessary buffer pointers, that used to render 3d model. This class is derived from NxUserRenderResource inetrface

Sources: ApexRenderer.cpp

class ApexFileStream

Implementation of file access interface. This class is needed to provide apex with ability to request different type of file resources, that can be stored in engine internal format (packaged, encrypted, etc.)

Derived from PxFileBuf class. Consult PhysX documentation for detailed description.

Sources: ApexFileStream.cpp, ApexFileStream.h

class obj_ApexDestructible

This class is derived from GameObject and represent game entity with attached apex actor. Each obj_ApexDestructible owns one instance of apex actor, update it and render it when corresponding methods a called by game manager.

Sources: obj_Apex.h, obj_Apex.cpp

Important methods

BOOL Update()

Update interal state of apex actor, by calling ApexActorBase::Update().

void AppendRenderables(RenderArray(&render_arrays)[rsCount], const r3dCamera& Cam)

Submit apex renderable chunks to the render pipeline.
Parameters:
rende_arrays – render arrays to submit renderable chunk into.
Cam – current camera.

void AppendShadowRenderables(RenderArray & rarr, const r3dCamera& Cam)

Submit apex renderable chunks into render array that responsible for shadow rendering. This allows to make apex objects debris behave like shadowcasters.
Parameters:
rarr – render array.
Cam – current camera.

BOOL OnCreate()

This function is called by game object manager to create object. Apex actor is created, and initialized with default parameters.

Return true if creation was successful.

void SetPosition(const r3dPoint3D& pos)

Set object position

Parameters:
pos – new object position.

void SetRotationVector(const r3dVector& Angles)

Set object rotation.

Parameters:
Angles – new object euler angles.

float DrawPropertyEditor(float scrx, float scry, float scrw, float scrh, const UClass* startClass, const GameObjects& selected)

This function executed only in level editor, while apex object rollout is active. It draws all necessary GUI configuration controls.

Parameters:
scrx – screen x coordinate that indicate drawing position.
scry - screen y coordinate that indicate drawing position.
scrw – screen width.
scrh – screen height.
startClass – ignored by this function.
selected – list of selected scene objects. Doesn't needed for apex configuration.

Return new screen y drawing position.

void ConvertToDestructible()

Convert internal apex object to destructible state.

void ConvertToPreview()

Convert internal apex object to preview state.

struct ApexActorParameters

Apex actor creation parameters structure.

Important methods and members

UString assetPath;

Path to the apex file to create actor from. It should be created by 3D studio max plugin.

D3DXMATRIX pose;

Initial pose of apex actor.

void *userData;

Pointer to user data, that will be stored with apex actor. We store pointer to the owner object (obj_ApexDestructible).

void SetPose(const r3dPoint3D &pos, const r3dPoint3D &rotVec);

Construct pose matrix using position and rotation vectors.

Parameters:
pos – position vector.
rotVec – euler angles vector.

Standalone helper functions

ApexActorBase * CreateApexActor(const ApexActorParameters &params, bool makePreview);

This function is used to create Apex preview actor or actual apex actor with given input parameters.

Parameters:
params – properly filled parameters structure.
makePreview – if object should be preview object pass true in this parameter.