Overview

To visualize objects in a world CryENGINE defines the concept of render nodes and render elements. Render nodes represent general objects in the 3D engine. Among other things they are used to build a hierarchy for visibility culling, allow physics interactions (optional) and rendering.

For actual rendering they add themselves to the renderer (with the help of render objects as you can see in the sample code below) passing an appropriate render element which implements the actual drawing of the object.

Creating a new render node

The following example render node is called **PrismObject**.

It needs to derive from **IRenderNode** defined in: `Code/CryEngine/CryCommon/IEntityRenderState.h`.

- Add the interface for IPrismObjectRenderNode to *CryEngine/CryCommon/IEntityRenderState.h* to make it publicly available

```
struct IPrismRenderNode : public IRenderNode
{
	...
};
```

- Add a new enum to the list of already defined render nodes in *CryEngine/CryCommon/IEntityRenderState.h*

```
enum EERType
{
	...
	eERType_PrismObject,
	...
};
```

- Add !PrismObjectRenderNode.h to !Cry3DEngine

```
#ifndef _PRISM_RENDERNODE_
#define _PRISM_RENDERNODE_

#pragma once

class CPrismRenderNode : public IPrismRenderNode, public Cry3DEngineBase
{
public:
         // interface IPrismRenderNode
         ...

         // interface IRenderNode
         virtual void SetMatrix(const Matrix34& mat);
         virtual EERType GetRenderNodeType();
         virtual const char* GetEntityClassName() const { return "PrismObject"; }
         virtual const char* GetName() const;
         virtual Vec3 GetPos(bool bWorldOnly = true) const;
         virtual bool Render(const SRendParams &rParam);
         virtual IPhysicalEntity* GetPhysics() const { return 0; }
         virtual void SetPhysics(IPhysicalEntity*) {}
         virtual void SetMaterial(IMaterial* pMat) { m_pMaterial = pMat; }
         virtual IMaterial* GetMaterial(Vec3* pHitPos = 0) { return m_pMaterial; }
         virtual float GetMaxViewDist();
         virtual void GetMemoryUsage(ICrySizer* pSizer);
         virtual const AABB GetBBox() const { return m_WSBBox; }
         virtual void SetBBox( const AABB& WSBBox ) { m_WSBBox = WSBBox; }

private:
         CPrismRenderNode();

private:
         ~CPrismRenderNode();

         AABB m_WSBBox;
         Matrix34 m_mat;
         _smart_ptr< IMaterial > m_pMaterial;
         CREPrismObject* m_pRE;
};

#endif // #ifndef _PRISM_RENDERNODE_
```

- Add !PrismObjectRenderNode.cpp to !Cry3DEngine

```
#include "StdAfx.h"
#include "PrismRenderNode.h"

CPrismRenderNode::CPrismRenderNode()
: m_WSBBox()
, m_mat()
, m_pMaterial(0)
, m_pRE(0)
{
         m_mat.SetIdentity();
         m_WSBBox = AABB(Vec3(-1, -1, -1), Vec3(1, 1, 1));
         m_pRE = (CREPrismObject*) GetRenderer()->EF_CreateRE(eDATA_PrismObject);
         m_dwRndFlags |= ERF_CASTSHADOWMAPS;
}

CPrismRenderNode::~CPrismRenderNode()
{
         if (m_pRE)
                  m_pRE->Release();

         Get3DEngine()->UnRegisterEntity(this);
         Get3DEngine()->FreeRenderNodeState(this);
```

```
}

void CPrismRenderNode::SetMatrix(const Matrix34& mat)
{
	Get3DEngine()->UnRegisterEntity(this);
	m_mat = mat;
	m_WSBBox.SetTransformedAABB(mat, AABB(Vec3(-1, -1, -1), Vec3(1, 1, 1)));
	Get3DEngine()->RegisterEntity(this);
}

EERType CPrismRenderNode::GetRenderNodeType()
{
	return eERType_PrismObject;
}

const char* CPrismRenderNode::GetName() const
{
	return "PrismObject";
}

Vec3 CPrismRenderNode::GetPos(bool bWorldOnly) const
{
	return m_mat.GetTranslation();
}

bool CPrismRenderNode::Render(const SRendParams& rParam)
{
	if(!m_pMaterial)
		return false;

	// create temp render node to submit this prism object to the renderer
	CRenderObject *pRO = GetRenderer()->EF_GetObject(true);

	// set basic render object properties
	pRO->m_II.m_Matrix = m_mat;
	pRO->m_ObjFlags |= FOB_TRANS_MASK;
	pRO->m_fSort = 0;
	pRO->m_fDistance = rParam.fDistance;

	// set render object properties
	m_pRE->m_center = m_mat.GetTranslation();

	if (pRO)
	{
		SShaderItem shaderItem(m_pMaterial->GetShaderItem(0));

		if (rParam.nTechniqueID > 0)
		    shaderItem.m_nTechnique = shaderItem.m_pShader->GetTechniqueID(
			shaderItem.m_nTechnique,rParam.nTechniqueID);

	       GetRenderer()->EF_AddEf(m_pRE, shaderItem, pRO, EFSLIST_GENERAL, 0);
	 }
	 return pRO != 0;
}

float CPrismRenderNode::GetMaxViewDist()
{
	return 1000.0f;
}

void CPrismRenderNode::GetMemoryUsage(ICrySizer* pSizer)
{
	SIZER_COMPONENT_NAME(pSizer, "PrismRenderNode");
	pSizer->AddObject(this, sizeof(*this));
}
```

- To allow client code to create an instance of the new render node extend the following function in *Code/CryEngine/Cry3DEngine/3DEngine.cpp*

```
...
#include "PrismRenderNode.h"
...
IRenderNode * C3DEngine::CreateRenderNode(EERType type)
{
	switch (type)
	{
	...
	case eERType_PrismObject:
	{
		IPrismRenderNode* pRenderNode = new CPrismRenderNode();
		return pRenderNode;
	}
	...
```