Adding multiple glow support(T3D 1.2)

This PDF describes how to add a secondary glow (we chose the "radial glow" effect as an example) to the Torque 3D project.

materialDefinition.h

```
bool mGlow[MAX_STAGES];
```

And below add:

```
bool mRadialGlow[MAX_STAGES];
```

materialDefinition.cpp

Locate Material::Material()

```
mGlow[i] = false;
```

blow it add:

```
mRadialGlow[i] = false;
```

Locate Material::initPersistFields()

```
addField("glow",                TypeBool,
Offset(mGlow, Material), MAX_STAGES);
```

below it add:

```
addField("radialglow",          TypeBool,
Offset(mRadialGlow, Material), MAX_STAGES);
```

processedMaterial.h

locate:

```
bool mGlow;
```

and below it add:

```
bool mRadialGlow;
```

A few lines below locate the state enumerator and replace with:

```
   enum
   {
      STATE_REFLECT = 1,
      STATE_TRANSLUCENT = 2,
      STATE_GLOW = 4,
      STATE_RADIALGLOW = 8,
      STATE_WIREFRAME = 16,
      STATE_MAX = 32,
```

```
   };
```

Find:

```
   bool hasGlow()
   {
      return mHasGlow;
   }
```

and below it add:

```
   bool hasRadialGlow()
   {
      return mHasRadialGlow;
   }
```

Locate :

```
/// If we glow
bool mHasGlow;
```

and add below :

```
bool mHasRadialGlow;
```

Find the stateblock constructor and replace with:

```
virtual void _initStateBlockTemplates(GFXStateBlockDesc&
stateTranslucent, GFXStateBlockDesc& stateGlow,
GFXStateBlockDesc& stateRadialGlow, GFXStateBlockDesc&
stateReflect);
```

processedMaterial.cpp

In RenderPassData::reset(), find

```
 mGlow = false;
```

and add:

```
 mRadialGlow = false;
```

In the constructor ProcessedMaterial::ProcessedMaterial()
find:

```
   mHasGlow( false ),
```

and add below:

```
   mHasRadialGlow( false ),
```

Locate _initStateBlockTemplates() add replace with:

```
void
ProcessedMaterial::_initStateBlockTemplates(GFXStateBlockDesc&
stateTranslucent, GFXStateBlockDesc&
```

```
stateGlow,GFXStateBlockDesc& stateRadialGlow,
GFXStateBlockDesc& stateReflect)
{
   // Translucency
   stateTranslucent.blendDefined = true;
   stateTranslucent.blendEnable = mMaterial-
>mTranslucentBlendOp != Material::None;
   _setBlendState(mMaterial->mTranslucentBlendOp,
stateTranslucent);
   stateTranslucent.zDefined = true;
   stateTranslucent.zWriteEnable = mMaterial-
>mTranslucentZWrite;
   stateTranslucent.alphaDefined = true;
   stateTranslucent.alphaTestEnable = mMaterial->mAlphaTest;
   stateTranslucent.alphaTestRef = mMaterial->mAlphaRef;
   stateTranslucent.alphaTestFunc = GFXCmpGreaterEqual;
   stateTranslucent.samplersDefined = true;
   stateTranslucent.samplers[0].textureColorOp =
GFXTOPModulate;
   stateTranslucent.samplers[0].alphaOp = GFXTOPModulate;
   stateTranslucent.samplers[0].alphaArg1 = GFXTATexture;
   stateTranslucent.samplers[0].alphaArg2 = GFXTADiffuse;

   // Glow
   stateGlow.zDefined = true;
   stateGlow.zWriteEnable = false;
   stateRadialGlow.zDefined = true;
   stateRadialGlow.zWriteEnable = false;

   // Reflect
   stateReflect.cullDefined = true;
   stateReflect.cullMode = mMaterial->mDoubleSided ?
GFXCullNone : GFXCullCW;
}
```

Now locate _initRenderStateStateBlocks() and replace with:

```
void ProcessedMaterial::_initRenderStateStateBlocks(
RenderPassData *rpd )
{
   GFXStateBlockDesc stateTranslucent;
   GFXStateBlockDesc stateGlow;
   GFXStateBlockDesc stateRadialGlow;
   GFXStateBlockDesc stateReflect;
   GFXStateBlockDesc statePass;

   _initStateBlockTemplates( stateTranslucent, stateGlow
,stateRadialGlow , stateReflect );
   _initPassStateBlock( rpd, statePass );
```

```
   // Ok, we've got our templates set up, let's combine them
together based on state and
   // create our state blocks.
   for (U32 i = 0; i < RenderPassData::STATE_MAX; i++)
   {
      GFXStateBlockDesc stateFinal;

      if (i & RenderPassData::STATE_REFLECT)
         stateFinal.addDesc(stateReflect);
      if (i & RenderPassData::STATE_TRANSLUCENT)
         stateFinal.addDesc(stateTranslucent);
      if (i & RenderPassData::STATE_GLOW)
         stateFinal.addDesc(stateGlow);
       if (i & RenderPassData::STATE_RADIALGLOW)
         stateFinal.addDesc(stateRadialGlow);

      stateFinal.addDesc(statePass);

      if (i & RenderPassData::STATE_WIREFRAME)
         stateFinal.fillMode = GFXFillWireframe;

      GFXStateBlockRef sb = GFX->createStateBlock(stateFinal);
      rpd->mRenderStates[i] = sb;
   }
}
```

Locate _getRenderStateIndex(),find:

```
   if ( sgData.binType == SceneGraphData::GlowBin )
      currState |= RenderPassData::STATE_GLOW;
```

add below:

```
   if ( sgData.binType == SceneGraphData::RadialGlowBin )
      currState |= RenderPassData::STATE_RADIALGLOW;
```

materials\sceneData.h

replace the bin enumerator:

```
   /// The special bin types.
   enum BinType
   {
      /// A render bin that isn't one of the
      /// special bins we care about.
      OtherBin = 0,

      /// The glow render bin.
      /// @see RenderGlowMgr
      GlowBin,
      RadialGlowBin,
```

```
      /// The prepass render bin.
      /// @RenderPrePassMgr
      PrePassBin,
   };
```

processedFFMaterial.cpp

locate ProcessedFFMaterial::_construct(),
find:

```
mHasGlow = false;
```

and below add:

```
mHasRadialGlow = false;
```

Locate ProcessedFFMaterial::_addPass(),
Find:

```
rpd.mGlow = false;
```

add below:

```
rpd.mRadialGlow = false;
```

matInstance.h

locate:

```
virtual bool hasGlow();
```

add below:

```
virtual bool hasRadialGlow();
```

baseMatInstance.h
locate:

```
virtual bool hasGlow() = 0;
```

add below:

```
virtual bool hasRadialGlow() = 0;
```

matInstance.cpp

locate:

```
bool MatInstance::hasGlow()
{
   if( mProcessedMaterial )
      return mProcessedMaterial->hasGlow();
   else
      return false;
}
```

and add below:

```
bool MatInstance::hasRadialGlow()
{
```

```
   if( mProcessedMaterial )
      return mProcessedMaterial->hasRadialGlow();
   else
      return false;
}
```

processedShaderMaterial.cpp

Locate: ProcessedShaderMaterial::_addPass(), find:

```
   rpd.mGlow |= mMaterial->mGlow[stageNum];
```

and add below:

```
   rpd.mRadialGlow |= mMaterial->mRadialGlow[stageNum];
```

a few lines below find:

```
   if( rpd.mGlow )
      mHasGlow = true;
```

and add:

```
   if( rpd.mRadialGlow )
      mHasRadialGlow = true;
```

a few lines below find and modify:

```
   if (  features.hasFeature( MFT_UseInstancing ) &&
         mMaxStages == 1 &&
         !mMaterial->mGlow[0] &&
         !mMaterial->mRadialGlow[0] &&
         shaderVersion >= 3.0f )
      fd.features.addFeature( MFT_UseInstancing );
```

materials\materialFeatureTypes.h

find:

```
DeclareFeatureType( MFT_GlowMask );
```

And add:

```
DeclareFeatureType( MFT_RadialGlowMask );
```

materials\materialFeatureTypes.cpp

find:

```
ImplementFeatureType( MFT_GlowMask, MFG_PostLighting, 1.0f,
true );
```

And add:

```
ImplementFeatureType( MFT_RadialGlowMask, MFG_PostLighting,
1.0f, true );
```

\shaderGen\hlsl\shaderGenHLSLInit.cpp

find:

```
   FEATUREMGR->registerFeature( MFT_GlowMask, new GlowMaskHLSL
);
```

And add:

```
   FEATUREMGR->registerFeature( MFT_RadialGlowMask, new
RadialGlowMaskHLSL );
```

\shaderGen\hlsl\shaderFeatureHLSL.h
below the glow declaration add:

```
class RadialGlowMaskHLSL : public ShaderFeatureHLSL
{
public:
   virtual void processPix(   Vector<ShaderComponent*>
&componentList,
                              const MaterialFeatureData &fd );

   virtual Material::BlendOp getBlendOp() { return
Material::None; }

   virtual String getName()
   {
      return "Glow Mask";
   }
};
```

\shaderGen\hlsl\shaderFeatureHLSL.cpp

below the glow shader feature add:

```
void RadialGlowMaskHLSL::processPix(
Vector<ShaderComponent*> &componentList,
                                 const MaterialFeatureData &fd
)
{
   output = NULL;

   // Get the output color... and make it black to mask out
   // glow passes rendered before us.
   //
   // The shader compiler will optimize out all the other
   // code above that doesn't contribute to the alpha mask.
   Var *color = (Var*)LangElement::find( "col" );
   if ( color )
      output = new GenOp( "   @.rgb = 0;\r\n", color );
}
```

renderInstance\renderPassManager.cpp
find:
#include "renderInstance/renderGlowMgr.h"
and add the directive
#include "renderInstance/renderRadialGlowMgr.h"

Now we have to dublicate the glow bin:
renderInstance\RenderRadialGlowMgr.h

```
#ifndef _RENDERRGLOWMGR_H_
#define _RENDERRGLOWMGR_H_

#ifndef _TEXTARGETBIN_MGR_H_
#include "renderInstance/renderTexTargetBinManager.h"
#endif


class PostEffect;


///
class RenderRadialGlowMgr : public RenderTexTargetBinManager
{
   typedef RenderTexTargetBinManager Parent;

public:

   RenderRadialGlowMgr();
   virtual ~RenderRadialGlowMgr();

   /// Returns the glow post effect.
   PostEffect* getGlowEffect();

   /// Returns true if the glow post effect is
   /// enabled and the glow buffer should be updated.
   bool isGlowEnabled();

   // RenderBinManager
   virtual void addElement( RenderInst *inst );
   virtual void render( SceneRenderState *state );

   // ConsoleObject
   DECLARE_CONOBJECT( RenderRadialGlowMgr );

protected:

   class GlowMaterialHook : public MatInstanceHook
   {
   public:

      GlowMaterialHook( BaseMatInstance *matInst );
```

```
      virtual ~GlowMaterialHook();

      virtual BaseMatInstance *getMatInstance() { return
mGlowMatInst; }

      virtual const MatInstanceHookType& getType() const {
return Type; }

      /// Our material hook type.
      static const MatInstanceHookType Type;

   protected:

      static void _overrideFeatures(   ProcessedMaterial *mat,
                                       U32 stageNum,
                                       MaterialFeatureData
&fd,
                                       const FeatureSet
&features );

      BaseMatInstance *mGlowMatInst;
   };

   SimObjectPtr<PostEffect> mGlowEffect;

};


#endif // _RENDERRGLOWMGR_H_
```

renderInstance\RenderRadialGlowMgr.cpp

```
#include "platform/platform.h"
#include "renderInstance/renderRadialGlowMgr.h"

#include "scene/sceneManager.h"
#include "scene/sceneRenderState.h"
#include "materials/sceneData.h"
#include "materials/matInstance.h"
#include "materials/materialFeatureTypes.h"
#include "materials/processedMaterial.h"
#include "postFx/postEffect.h"
#include "gfx/gfxTransformSaver.h"
#include "gfx/gfxDebugEvent.h"
#include "math/util/matrixSet.h"

IMPLEMENT_CONOBJECT( RenderRadialGlowMgr );


ConsoleDocClass( RenderRadialGlowMgr,
```

```
   "@brief A render bin for the glow pass.\n\n"
   "When the glow buffer PostEffect is enabled this bin
gathers mesh render "
   "instances with glow materials and renders them to the
#glowbuffer offscreen "
   "render target.\n\n"
   "This render target is then used by the 'GlowPostFx'
PostEffect to blur and "
   "render the glowing portions of the screen.\n\n"
   "@ingroup RenderBin\n" );

const MatInstanceHookType
RenderRadialGlowMgr::GlowMaterialHook::Type( "RadialGlow" );


RenderRadialGlowMgr::GlowMaterialHook::GlowMaterialHook(
BaseMatInstance *matInst )
   : mGlowMatInst( NULL )
{
   mGlowMatInst = (MatInstance*)matInst->getMaterial()-
>createMatInstance();
   mGlowMatInst->getFeaturesDelegate().bind(
&GlowMaterialHook::_overrideFeatures );
   mGlowMatInst->init(  matInst->getRequestedFeatures(),
                        matInst->getVertexFormat() );
}

RenderRadialGlowMgr::GlowMaterialHook::~GlowMaterialHook()
{
   SAFE_DELETE( mGlowMatInst );
}

void RenderRadialGlowMgr::GlowMaterialHook::_overrideFeatures(
ProcessedMaterial *mat,
                                                        U32
stageNum,

MaterialFeatureData &fd,
                                                        const
FeatureSet &features )
{
   // If this isn't a glow pass... then add the glow mask
feature.
   if (  mat->getMaterial() &&
         !mat->getMaterial()->mRadialGlow[stageNum] )
      fd.features.addFeature( MFT_RadialGlowMask );

   // Don't allow fog or HDR encoding on
   // the glow materials.
   fd.features.removeFeature( MFT_Fog );
   fd.features.removeFeature( MFT_HDROut );
```

```
}

RenderRadialGlowMgr::RenderRadialGlowMgr()
   : RenderTexTargetBinManager(  RenderPassManager::RIT_Mesh,
                                 1.0f,
                                 1.0f,
                                 GFXFormatR8G8B8A8,
                                 Point2I( 512, 512 ) )
{
   notifyType( RenderPassManager::RIT_Interior );
   notifyType( RenderPassManager::RIT_Decal );
   notifyType( RenderPassManager::RIT_Translucent );

   mNamedTarget.registerWithName( "radialglowbuffer" );
   mTargetSizeType = WindowSize;
}

RenderRadialGlowMgr::~RenderRadialGlowMgr()
{
}

PostEffect* RenderRadialGlowMgr::getGlowEffect()
{
   if ( !mGlowEffect )
      mGlowEffect = dynamic_cast<PostEffect*>(
Sim::findObject( "RadialGlowPostFx" ) );

   return mGlowEffect;
}

bool RenderRadialGlowMgr::isGlowEnabled()
{
   return getGlowEffect() && getGlowEffect()->isEnabled();
}

void RenderRadialGlowMgr::addElement( RenderInst *inst )
{
   // Skip out if we don't have the glow post
   // effect enabled at this time.
   if ( !isGlowEnabled() )
      return;

   // TODO: We need to get the scene state here in a more
reliable
   // manner so we can skip glow in a non-diffuse render pass.
   //if ( !mParentManager->getSceneManager()->getSceneState()-
>isDiffusePass() )
      //return RenderBinManager::arSkipped;

   // Skip it if we don't have a glowing material.
   BaseMatInstance *matInst = getMaterial( inst );
```

```
   if ( !matInst || !matInst->hasRadialGlow() )
      return;

   internalAddElement(inst);
}

void RenderRadialGlowMgr::render( SceneRenderState *state )
{
   PROFILE_SCOPE( RenderGlowMgr_Render );

   if ( !isGlowEnabled() )
      return;

   const U32 binSize = mElementList.size();

   // If this is a non-diffuse pass or we have no objects to
   // render then tell the effect to skip rendering.
   if ( !state->isDiffusePass() || binSize == 0 )
   {
      getGlowEffect()->setSkip( true );
      return;
   }

   GFXDEBUGEVENT_SCOPE( RenderGlowMgr_Render, ColorI::GREEN );

   GFXTransformSaver saver;

   // Tell the superclass we're about to render, preserve
contents
   const bool isRenderingToTarget = _onPreRender( state, true
);

   // Clear all the buffers to black.
   GFX->clear( GFXClearTarget, ColorI::BLACK, 1.0f, 0);

   // Restore transforms
   MatrixSet &matrixSet = getRenderPass()->getMatrixSet();
   matrixSet.restoreSceneViewProjection();

   // init loop data
   SceneData sgData;
   sgData.init( state, SceneData::GlowBin );

   for( U32 j=0; j<binSize; )
   {
      MeshRenderInst *ri =
static_cast<MeshRenderInst*>(mElementList[j].inst);

      setupSGData( ri, sgData );

      BaseMatInstance *mat = ri->matInst;
```

```
      GlowMaterialHook *hook = mat-
>getHook<GlowMaterialHook>();
      if ( !hook )
      {
         hook = new GlowMaterialHook( ri->matInst );
         ri->matInst->addHook( hook );
      }
      BaseMatInstance *glowMat = hook->getMatInstance();

      U32 matListEnd = j;

      while( glowMat && glowMat->setupPass( state, sgData ) )
      {
         U32 a;
         for( a=j; a<binSize; a++ )
         {
            MeshRenderInst *passRI =
static_cast<MeshRenderInst*>(mElementList[a].inst);

            if ( newPassNeeded( ri, passRI ) )
               break;

            matrixSet.setWorld(*passRI->objectToWorld);
            matrixSet.setView(*passRI->worldToCamera);
            matrixSet.setProjection(*passRI->projection);
            glowMat->setTransforms(matrixSet, state);

            glowMat->setSceneInfo(state, sgData);
            glowMat->setBuffers(passRI->vertBuff, passRI-
>primBuff);

            if ( passRI->prim )
               GFX->drawPrimitive( *passRI->prim );
            else
               GFX->drawPrimitive( passRI->primBuffIndex );
         }
         matListEnd = a;
         setupSGData( ri, sgData );
      }

      // force increment if none happened, otherwise go to end
of batch
      j = ( j == matListEnd ) ? j+1 : matListEnd;
   }

   // Finish up.
   if ( isRenderingToTarget )
      _onPostRender();

   // Make sure the effect is gonna render.
   getGlowEffect()->setSkip( false );
```

```
}
```

Ok,now this bin will render the radial masked meshes to a separate buffer called 'radialglowbuffer' !
Why we need to create a secondary bin ?
Because we need to know which meshes to be rendered and where to apply the post effect.

Now we go to:
game\core\scripts\client\renderManager.cs

And we add the new bin below the old glow bin:

```
// Note that the GlowPostFx is triggered after this bin.
DiffuseRenderPassManager.addManager( new
RenderGlowMgr(GlowBin) { renderOrder = 1.5; processAddOrder =
1.5; } );
DiffuseRenderPassManager.addManager( new
RenderRadialGlowMgr(RadialGlowBin) { renderOrder = 1.5;
processAddOrder = 1.5; } );
```

game\core\scripts\client\postFX\glow.cs
We add this script at the bottom of the file:

```
singleton ShaderData( PFX_RadialGlowShader )
{
   DXVertexShaderFile    = "shaders/common/postFx/
Liman_vol2_glow_effects/radialGlowV.hlsl";
   DXPixelShaderFile     = "shaders/common/postFx/
Liman_vol2_glow_effects/radialGlowP.hlsl";

   samplerNames[0] = "$diffuseMap";
   pixVersion = 3.0;
};

singleton GFXStateBlockData( PFX_GlowCombineStateBlock2 :
PFX_DefaultStateBlock )
{
   // Use alpha test to save some fillrate
   // on the non-glowing areas of the scene.
   alphaDefined = true;
   alphaTestEnable = true;
   alphaTestRef = 1;
   alphaTestFunc = GFXCmpGreaterEqual;

   // Do a one to one blend.
   blendDefined = true;
   blendEnable = true;
   blendSrc = GFXBlendOne;
```

```
   blendDest = GFXBlendOne;
};
singleton PostEffect( RadialGlowPostFx )
{
   // Do not allow the glow effect to work in reflection
   // passes by default so we don't do the extra drawing.
   allowReflectPass = false;

   renderTime = "PFXAfterDiffuse";
   renderBin = "RadialGlowBin";
   renderPriority = 1;

   // First we down sample the glow buffer.
   shader = PFX_PassthruShader;
   stateBlock = PFX_DefaultStateBlock;
   texture[0] = "#radialglowbuffer";
   target = "$outTex";
   targetScale = "0.5 0.5";

   isEnabled = true;

   new PostEffect()
   {
      shader = PFX_RadialGlowShader;
      stateBlock = PFX_DefaultStateBlock;
      texture[0] = "$inTex";
      target = "$outTex";
   };

   new PostEffect()
   {
      shader = PFX_RadialGlowShader;
      stateBlock = PFX_DefaultStateBlock;
      texture[0] = "$inTex";
      target = "$outTex";
   };

   // Upsample and combine with the back buffer.
   new PostEffect()
   {
      shader = PFX_PassthruShader;
      stateBlock = PFX_GlowCombineStateBlock2;
      texture[0] = "$inTex";
      target = "$backBuffer";
   };
};
```

Now we are ready to use both of the bins - in material.cs use glow and radialglow.

```
singleton Material(ourMaterial)
{ ..
    radialglow[0] = true; // turn on radialglow
    glow[0] = false; // turn off glow
};
```

Pay attention to how we use the new render bin in our new post effect.
Additional notes : dublicating the glow shader feature is not necessarily! We did not have enought time to experiment and optimize this resource.
You can use different priorities for different glow effects.

This resource describes how to add a second glow effect support (in this case “radial glow”). Adding more glow effects can be achieved according to this resource.Have in mind that every additional glow bin will hurt your framerate. Usually glow meshes are rendered three times (prepass,renderMeshbin,glowbin)! It means if you render glow meshes into several glow bins,they will be rendered 3 or more times.