آشنايي با HLSL

نويسنده :ميثم سلطانی

WWW.Persian-Designers.COM

meysamsoltany@gmail.com

به طور خلاصه shader برنامه کو چکي است که روي GPU اجرا مي شود . که با قسمتي از fixed function pipeline جايگزين مي شود . با اين جايگزيني ساخت انواع افکتهاي گرافيکي ميسر مي شود و به fixed function هاي پيش تعريف شده نياز نيست . شيدر براي مشخص کردن خصوصيات نهايي اشيا يا تصاوير به کار مي رود. reflection, texture mapping, diffusion, light absorption , displacement, surface shadowing, refraction و post-processing نمونه هايي از اين افکتها هستند

استفاده از برنامه هاي مختلف در تکنولوژي هاي مختلف دو نوع شيدر بر اساس استفاده از کد يا عدم استفاده از کد را با نام production rendering, real-time rendering بوجود آورده است . شيدرهاي production rendering اولين بار در شرکت Pixar با نام RenderMan در ساخت فيلمهاي سينمايي بوجود آمد . Glato shading language هم توسط nVidia براي برنامه Glato ساخته شد .

RealTime Shaders

تا اين قسمت يک ديد کلي از معني شيدر و انواع شيدر بدست آورديم . حال به شيدر هاي Real Time مي پردازيم . از اين قسمت به بعد منظور از شيدر، نوع RealTime است . در واقع دو نوع برنامه شيدر وجود دارد ، اگر چه فهرست خصوصيات اين دو نوع برنامه مانند هم است ولي اهداف محاسباتي مختف باعث چنين اختلاف و محدوديتي شده اند .

Vertex shaders

Vertex shader ها روي همه vertex ها با استفاده از يک vertex processor اجرا مي شوند . vertex shader ها متدي را براي محاسبه تبديلات فضاي برداري و ديگر محاسبات تعريف مي کنند .

Pixel shaders

اين نوع شيدرها براي محاسبه خصوصياتي استفاده مي شوند که در بيشتر موارد به عنوان رنگ vertex تلقي مي شوند. اين شيدرها رو هر پيکسل با استفاده از يک pixel processor اجرا مي شود . اين نوع شيدر نسبت به vertex shader ها به قدرت محاسباتي بيشتري نياز دارد.

زبان هاي شيدر نويسي

GLSL

زبان شيدر نويسي OpenGL يا GLSL يا glslang است. که يک زبان سطح بالا مي باشد.

Cg

C براي g ، يک زبان شيدر سطح بالاست که توسط nVidia براي شيدر نويسي ساخته شده است . Cg بر اساس زبان برنامه نويسي C ساخته شده است و اگر چه از لحاظ syntax شبيه C است ، بعضي از خصوصيات C تغيير يافته است . و انواع داده جديدي تعريف شده تا Cg براي شيدر نويسي مناسب تر باشد . اين زبان طراحي شده تا مجتمع کردن pipeline ها ساده و کارا باشد . استفاده هاي اوليه از Cg ، بيشتر محدود کننده بود زيرا با پيشرفتهاي زياد سخت افزار ، نتوانسته بود پيشرفت کند

HLSL

در DirectX 8.x شيدر ها با زبان سطح پايين اسمبلي نوشته مي شدند . خوشبختانه ديگر مجبور به استفاده از اين زبان نيستيم زيرا DirectX 9 زبان سطح بالاي شيدري رو ارايه کرده است که مي توانيم از آن براي نوشتن شيدر استفاده کنيم . استفاده از HLSL بجاي اسمبلي در نوشتن شيدر همان مزاياي برنامه نويسي با C++ نسبت به اسمبلي مانند سرعت بيشتر ، خوانايي و توليد کد اسمبلي کاراتر توسط کامپايلر را داراست . همچنين زبان HLSL شباهت زيادي به C, C++ دارد بنابراين ياد گيري آن نيازمند زمان کمتري است . از ديگر خصوصيات HLSL استفاده از REF Device در حالتي است که GPU شيدر مورد نظر را پشتيباني نمي کند

نوشتن يک شيدر به زبان HLSL

مي توان شيدر را به صورت يک رشته وارد برنامه کرد ولي بهتر است که کد شيدر از کد برنامه جدا باشد پس مي توانيم از يک ويرايشگر ساده مثل notepad و يا در موارد حرفه ای تر از نرم افزار render Monkey استفاده کنيم . به عنوان اولين برنامه يک Vertex shader مي نويسيم .

```
// Global variable to store a combined view and projection transformation matrix ,we
initialize this
```

```
// variable from application
matrix ViewProjMatrix;

// Initialize a global blue color vector
vector blue = {0.0 ,0.0 ,1.0 ,1.0};

//structures

// Input structure describes the vertex that is input to the shader. Here the input
vertex contains a
// position component only

struct VS_INPUT
{
    vector position : POSITION;
};

// Output structure describes the vertex that is output from shader. Here the output
vertex contains a
// position and color component


struct VS_OUTPUT
{
    vector position : POSITION;
    vector diffuse  : COLOR;
};

// Main entry point observe the main function receives a copy of the input vertex through
its parameters
// and returns a copy of the output vertex it compute

VS_OUTPUT Main(VS_INPUT input)
{
    // Zero out members of output
    VS_OUTPUT output = (VS_OUTPUT)0;

    // transform to view space and projection
    outpot.position = mul(input.position,ViewProjMatrix);

    // Set vertex  deiffuse color to blue
    output.diffuse = blue;

    // return projected and colored vertex
    return output;
}
```

متغيرهاي عمومي

ابتدا دو متغير عمومي تعريف شده است

```
matrix ViewProjMatrix;
vector blue = {0.0 ,0.0 ,1.0 ,1.0};
```

اواين متغير از جنس matrx است كه يک ماتريس ۴*۴ مي باشد . اين ماتريس تركيبي از ماتريس view و projection است . بنابراين نشاندهنده هر دو تبديل است . دقت كنيد كه در كد شيدر ، اين متغير را مقدار دهي نمي كنيم زيرا اين كار توسط كد برنامه انجام مي شود . ارتباط بين كد برنامه و كد شيدر هميشه لازم خواهد بود .
متغير دوم blue از جنس vector است . كه يک بردار ۴ بعدي مي باشد . به سادگي مولفه هاي ان را بر اساس مدل RGBA پر مي كنيم .

ساختارهاي ورودي و خروجي

بعد از تعريف متغيرهاي عمومي دو ساختار مخصوص به نامهاي input , output تعريف كرده ايم . براي vertex shader اين متغيرها متغيرهاي ورودي و خروجي vertex ها را مشخص مي كنند . در اين مثال vertex ورودي به شيدر فقط شامل يک مولفه مكان است . vertex خروجي شامل مولفه مكان و يک مولفه رنگ است . حال به بررسي اين مولفه ها مي پردازيم .

علامت : در تعريف يک مولفه مشخص كننده نحوه استفاده از يک متغير است (مانند flexible vertex format (FVF) در ساختار vertex) مثلا در VS_INPUT ، متغير `vector position : POSITION` را داريم . نماد `POSITION` : نشاندهنده اين است كه متغير position براي تعريف مكان يک vertex بكار مي رود . مثال ديگر در VS_OUTPUT ، `vector diffuse : COLOR` است . نماد : `COLOR` به اين معناست كه متغير diffuse براي تعريف رنگ vertex بكار مي رود .

تابع entry point

همانند برنامه هاي C++ هر شيدر داراي يک تابع entry است . در اين مثال تابع entry point ، Main است . اگرچه اين نام اجباري نيست . بنابراين نام اين تابع مي تواند هر نام قابل قبولي باشد . اين تابع بايد داراي يک مقدار ورودي باشد که vertex ورودي را وارد شيدر مي کند و همچنين اين تابع بايد مقدار خروجي را برگرداند

```
VS_OUTPUT Main(VS_INPUT input)
{
        ...
}
```

در واقع تعريف ساختارهاي ورودي و خروجي اجباري نيست . مثلا ممکن است يک entry point مانند زير باشد :

```
float4 Main (in float2 base : TEXCORD0 ,
             in float2 spot  : TEXCORD1,
             in float2 text : TEXCORD2) : COLOR
{
..
}
```

بدنه تابع entry point مقدار خروجي را از روي مقدار ورودي محاسبه مي کند . شيدر فوق به سادگي vertex ورودي را به فضاي view و projection تبديل مي کند و رنگ vertex را آبي مي کند . ابتدا يک متغير output تعريف کرده و آنرا با ۰ مقداردهي مي کنيم .

```
VS_OUTPUT output = (VS_OUTPUT)0;
```

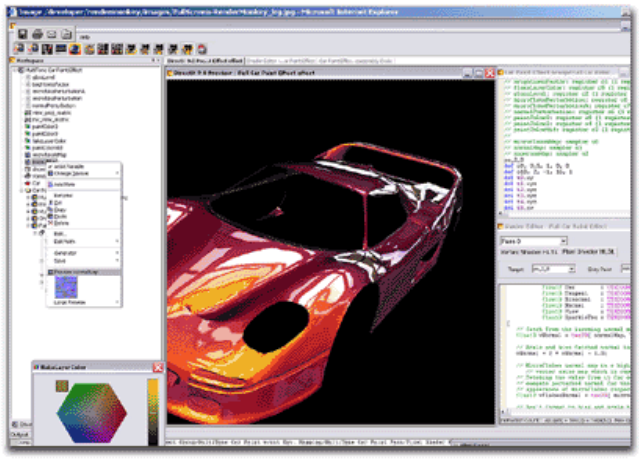
سپس شيدر مکان vertex را با متغير ViewProjMatrix و استفاده از mul تبديل مي کند . که يک تابع built-in است . که عمل ضرب vector در ماتريس و ماتريس در ماتريس را داراست . نتيجه شرب را در مولفه position مقدار خروجي ذخيره مي مي کنيم

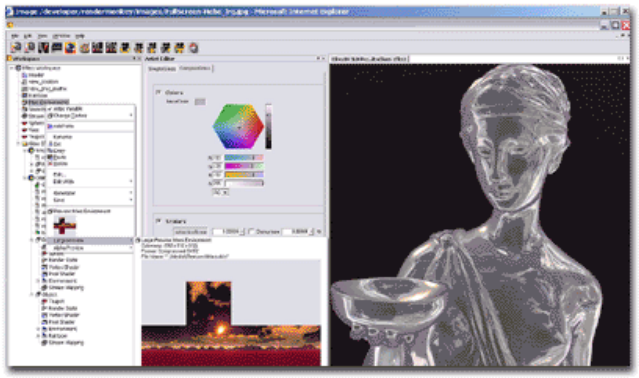
```
outpot.position = mul(input.position,ViewProjMatrix);
```

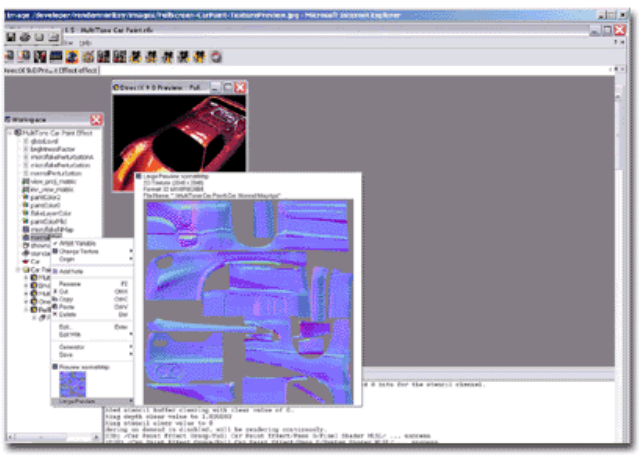
حال مقدار diffuse color خروجي را برابر blue قرار مي دهيم . و در خط آخر مقدار را بر مي گردانيم .

```
return output;
```

آشنايي با RenderMonkey







این برنامه یک محیط قوي براي نوشتن کدهاي شیدر است که توسط شرکت ATI ساخته شده است . با توجه به فعالیتهایی که شرکت nVidia و Microsoft در ساخت زبانهای شیدر نویسی داشته اند حتما انتظار دارید که شرکت ATI که رقیب nVidia در ساخت کارتهای گرافیکی است زبان مشابهی را طراحی و در کارتهای ATI استفاده کند . در سایت ATI نام بازی ساز های معروفی به چشم می خورد که از ATI برای ساخت این نرم افزار قدرتمند تشکر کرده اند . این محیط دارای ویژگیهای Syntax Coloring و Intellisense است . ATI می تواند الگوی مناسبی برای طیف وسیعی از فعالان صنعت بازی باشد که توجه خود را تنها به یک آیتم خاص معطوف نکنند .

ساده ترین مثال در rendermonkey نیاز به

۱. ماتریس projection and view
۲. vertex data stream که با vertex shader input stream مرتبط است
۳. یک مدل که باید نمایش داده شود
۴. گروهي از effect ها که شامل یک یا چند افکت است و هر افکت شامل یک یا چند render pass است

rendermonky شامل مجموعه اي از متغیرهاي پیش تعریف شده است . یکي از این متغیرها ماتریس view and projection است که view_proj_matrix نام دارد . براي اضافه کردن این متغیر در فضاي کاري از منوي ظاهر شده add variable را انتخاب و در دیالوگ ظاهر شده نوع matrix را انتخاب کنید. چند نوع متغیر دیگر مانند scalar,vector,matrix,texture,cube map,volume texture,color هم قابل انتخاب هستند

پس از انتخاب نوع متغیر در قسمت name نام view_proj_matrix را وارد کنید و دکمه OK را بفشارید

حال باید اطلاعات vertex را به vertex shader input stream متصل کنیم . به این منظور در فضاي کاري از منوي ظاهر شده گزینه add stream mapping را انتخاب کنید . حال روي stream mapping در پروژه دابل کلیک کنید تا دیالوگ مربوط ظاهر شود .براي اضافه کردن کانال = (vertexshader input register + usage + vertex stream) دکمه add channel را بفشارید

براي اضافه کردن مدل از فایل rendermonkey فایلهاي 3ds را پشتیباني مي کند . یک مدل با انتخاب گزینه add model از منو و سپس دبل کلیک model در پروژه لود مي شود .

در آخرین مرحله باید یک effect group ساخته شود . به این منظور گزینه add effect group را از منو اتخاب کنید . حال یک effect group پیش فرض ساخته شد . که شامل یک افکت ساده هست که داراي یک vertex shader , pixel shader ساده در یک render pass است . پس از اضافه کردن effect group مدل ما ظاهر مي شود . اگر تعداد افکتها بیش از یکي باشد مي توانیم افکتهاي مختلف را روشن و خاموش کنیم با انتخاب کردن افکت با راست کلیک ، افکت فعال مي شود. همانطور که گفته شد هر افکت مي تواند یک یا چند pass ذاشته باشد که با رایت کلیک enable\disable مي شود. در هر render pass ، node هاي model و stream mapping همان short cut به مدل و stream map اصلي هستند . vs node شامل vertex shader و ps node شامل pixel shader است .با دابل کلیک کردن vs ، کد در HLSL Editor باز مي شود.