

DirectX

۸۵۱۸۳۹

شهباز یعقوبی

پاییز ۱۳۸۶

فهرست.....	صفحه
مقدمه	۲
Direct3D.....	۲
سیستم مختصات 3D.....	۳
عناصر اولیه 3D.....	۴
ماتریس ها.....	۵
تبدیلات.....	۵
دوران.....	۶
آماده سازی محیط C# برای نویسی DIRECTX.....	۶
دوران در DirectX.....	۹
نور پردازی.....	۹
انواع منبع نور.....	۹
Vertex Buffers از استفاده.....	۱۰
بافتها.....	۱۱
مش ها.....	۱۳
استفاده از ماتریال ها و نور پردازی.....	۱۴
انواع دیگر مش.....	۱۵
منابع.....	۱۶

مقدمه

شرکت مایکروسافت با معرفی سیستم عامل ویندوز یک تحول عظیم را در تولید نرم افزارها ایجاد کرد . با فراگیر شدن سیستم عامل ویندوز اکثر شرکت های نرم افزاری تصمیم به انتقال نرم افزارهای خود به این سیستم عامل گرفتند . سیستم عامل ویندوز یک بستر خوب را برای برنامه های تجاری ایجاد کرده بود اما به نرم افزارها اجازه دسترسی مستقیم به سخت افزار را نمی داد . در نتیجه نرم افزارهای چند رسانه ای نظیر بازیها که نیاز به دسترسی سطح پایین به سخت افزار داشتند با مشکل مواجه شدند ، زیرا زیر سیستم ویندوز به گونه ای طراحی شده بود که به نرم افزارها اجازه چنین دسترسی را نمی داد و همچنین توابع ویندوز نیز نمی توانستند جوابگوی نیازهای این نرم افزارها باشند . شرکت مایکروسافت با توجه به اینکه نیاز فوق غیر قابل چشم پوشی بود ، یک محیط برنامه نویسی جدید به نام **Direct** را برای تولیدکنندگان چنین نرم افزارهایی ایجاد کرد . **Direct** اولین بار در سال ۱۹۹۵ معرفی شد و به یک استاندارد جهت برنامه نویسی چندرسانه ای در ویندوز تبدیل گردید . در واقع **Direct** از یک سری توابع برنامه نویسی **API** تشکیل شده است که به تولید کنندگان اجازه دسترسی به امکانات ویژه سخت افزارها را می دهد و این امکان را ایجاد می کند که از سخت افزار با حداکثر توانایی آن استفاده کند . امروزه **Direct** قلب اکثر برنامه های چند رسانه ای می باشد . **Direct** به برنامه ها اجازه می دهد که از امکانات پیشرفته سخت افزارها نظیر شتاب دهنده های گرافیک **3D** و کارتهای صدا دسترسی پیدا کنند .

DirectX دارای منابعی می باشد که در جدول زیر آمده است. که ما در مورد قسمت **Direct3D** بحث خواهیم کرد.

Microsoft.DirectX	شامل تمام دستورات معمولی، Parent namespace
Microsoft.DirectX.Direct3D	شامل توابع API برای رسم های سه بعدی میباشد
Microsoft.DirectX.DirectDraw	شامل توابع رسم گرافیکی
Microsoft.DirectX.DirectPlay	شامل API مربوط به شبکه
Microsoft.DirectX.DirectSound	شامل API مربوط به صدا
Microsoft.DirectX.DirectInput	شامل API مربوط به دستگاههای ورودی
Microsoft.DirectX.AudioVideoPlayback	Simple audio and video playback API.
Microsoft.DirectX.Diagnostics	شامل API مربوط به خطاها

Direct3D:

Direct3D دو حالت کاری دارد : حالت بلادرنگ و حالت ابقایی . حالت ابقایی شامل API های سطح پایین **3D** می باشد و در نتیجه وسیله خوبی برای تولید کنندگانی که نیاز به تولید بازی ها و سایر برنامه های چند رسانه ای سریع الاجرا دارند . حالت بلادرنگ یک روش غیر وابسته به دستگاه جهت ارتباط برنامه ها با سخت افزار شتاب دهنده گرافیک می باشد . توجه کنید که حالت ابقایی **Direct3D** بر روی حالت بلادرنگ **Direct3D** ایجاد شده است .

بعضی از قسمتهای پیشرفته **Direct3D** به شرح زیر می باشد :

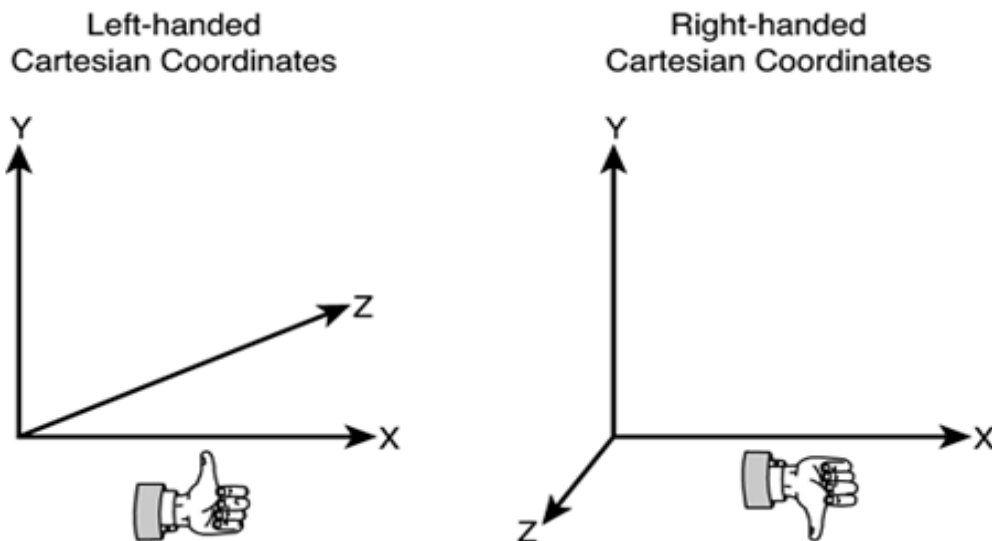
۱- بافر کردن عمق به صورت سوچینگ

- ۲- سایه گذاری تخت و گراد
- ۳- منابع نور چند تایی و انواع مختلفی از نور چندتایی
- ۴- پشتیبانی کامل از ماتریال ها و بافتها
- ۵- شبیه سازی نرم افزار قوی
- ۶- تبدیلات و دوران

سیستم مختصات 3D:

عموما برنامه های گرافیکی 3D از دو نوع سیستم مختصات استفاده می کنند: دست چپ و دست راست. در هر دو سیستم مختصات محور X مثبت در قسمت راست و محور Y در بالا قرار دارد و شما می توانید جهت محور Z را با قرار دادن یکی از انگشتان دست راست یا چپ خود بر روی محور X های مثبت و سپس چرخش آن در جهت محور Y های مثبت به دست آورید. پس از این کار جهت انگشت شست به طرف بیرون و یا به طرف داخل صفحه می باشد، که جهت محور Z مثبت را در آن سیستم مختصات نشان می دهد.

مطابق شکل زیر:



شکل ۱-۱

Direct3D از سیستم مختصات دست چپ استفاده می کند. اگر شما بر روی برنامه ای کار می کنید که براساس سیستم مختصاتی دست راست می باشد، باید دو تغییر را بر روی داده هایی که به Direct3D انتقال می دهید انجام دهید.

- (۱) معکوس کردن نظم رئوس سه ضلعی و در نتیجه تغییر آن به حالت ساعت گرد. در واقع اگر رئوس V_0, V_1, V_2 آنها را به صورت V_0, V_2, V_1 به Direct3D منتقل کنید.
- (۲) از ماتریس نمایش مقیاس بندی با -1 در جهت Z استفاده کنید.

عملیات های اصلی که بر روی یک شی در سیستم مختصات 3D انجام می شود عبارتند از: تبدیلات، دوران و تغییر مقیاس. شما می توانید از ترکیب این عملیاتها برای ایجاد ماتریس انتقال استفاده کنید.

عناصر اولیه 3D:

یکی از مجموعه عناصر 3D رئوس می باشند که موجودیت یک شی 3D را شکل می دهند. ساده ترین عنصر اولیه مجموعه ای از نقاط بر روی سیستم مختصات 3D می باشد که به آن لیست نقاط می گویند. اغلب عناصر 3D را چند ضلعی ها تشکیل می دهند. چند ضلعی یک شکل 3D بسته است که حداقل از سه راس ایجاد شده باشد. از ساده ترین چند ضلعی مثلث است. 3D از مثلث ها برای ایجاد خیلی از چند ضلعی ها استفاده می کند، زیرا هر سه راس موجود در مثلث همیشه به صورت همانی می باشند. همچنین می توانی از مثلث ها برای ایجاد تصاویر خمیده استفاده کنید مانند ایجاد یک کره.

ماتریس ها:

یک ماتریس در 3D به وسیله یک آرایه 4×4 نمایش داده می شود.

تبدیلات:

در برنامه هایی که به صورت 3D عمل می کنند شما می توانید از تبدیلات جهت اعمال زیر استفاده کنید:

(۱) تعیین صریح موقعیت یک شی وابسته به شی دیگر

(۲) دوران و تغییر در اندازه اشیا

(۳) تغییر موقعیت نمایش، جهت و پرسپکتیوها

شما می توانید ماتریس هایی که این عملیات را انجام می دهند با هم ترکیب کنید تا ماتریسی ایجاد کنید که کلیه این عملیات را با هم انجام دهد.

دوران:

ماتریس زیر نقاط X, Y, Z را حول محور X دوران داده و نقاط جدید X', Y', Z' را ایجاد می کند.

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos q & \sin q & 0 \\ 0 & -\sin q & \cos q & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ماتریس زیر نقاط X, Y, Z را حول محور Y دوران داده و نقاط جدید X', Y', Z' را ایجاد می کند.

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} \cos q & 0 & -\sin q & 0 \\ 0 & 1 & 0 & 0 \\ \sin q & 0 & \cos q & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ماتریس زیر نقاط X, Y, Z را حول محور Z دوران داده و نقاط جدید X', Y', Z' را ایجاد می کند.

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} \cos q & \sin q & 0 & 0 \\ -\sin q & \cos q & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ما برنامه نویسی DirectX را در زبانهای مختلف از جمله C#, C++ می توانیم انجام دهیم که ما در محیط NET. توسط زبان C# این کار را توضیح میدهیم.

آماده سازی محیط C# برای نویسی DirectX:

در ابتدا ما نیاز داریم که DirectX 9.0 SDK را نصب کنیم سپس اضافه کردن رفرنسهای زیر به منابع پروژه خود.

```
Using Microsoft.DirectX;
Using Microsoft.DirectX.Direct3D;
```

بعد از اضافه کردن این منابع به پروژه ما میتوانیم شروع به برنامه نویسی کنیم DirectX همچنین دارای منابع دیگری که در جدول بالا آمده است.

ریشه تمام رسم ها در Direct3D کلاس DEVICE است شما می توانید فکر کنید این کلاس مشابه دستگاه کارت گرافیکی شما است. لذا برای شروع کار اولین کار ایجاد یک شی از نوع کلاس DEVICE می باشد کلاس DEVICE شامل سه نوع سازنده است که ما برحسب نیاز در هنگام ایجاد شی یکی از آنها را فراخوانی میکنیم.

```
private Device device = null;
```

ما برای ایجاد و آماده سازی device نیاز به یک سری مقدار دهی های اولیه داریم لذا بهتر است این مقدار دهی را در درون یک تابع انجام دهیم.

```
public void InitializeGraphics()
{
    // Set our presentation parameters
    PresentParameters presentParams = new PresentParameters();

    presentParams.Windowed = true;
    presentParams.SwapEffect = SwapEffect.Discard;

    // Create our device
    device = new Device(0, DeviceType.Hardware, this,
        CreateFlags.SoftwareVertexProcessing, presentParams);
}
```

منبع: [Tom Miller](#) By Managed DirectX® 9 Kick Start: Graphics and Game Programming

در اینجا به شرح مختصری از پارامترهای سازنده **Device** میپردازیم که در هنگام **new** کردن باید به سازنده ارسال شوند.

۱- **Adapter**: اولین پارامتری که ارسال می شود مشخص کننده کارت گرافیکی که ما می خواهیم استفاده کنیم می باشد هر دستگاه در هر کامپیوتر دارای یک کد شناسای منحصر به فرد می باشد که این پارامتر به صورت پیش فرض 0 در نظر می گیریم.

۲- **DeviceType**: نوع **device** که ما می خواهیم ایجاد کنیم را بیان میکند که **DeviceType.Hardware** به معنی این است که ما می خواهیم استفاده کنیم از سخت افزار و **DeviceType.refrence** به این معنی است که ما می خواهیم از شبیه ساز نرم افزاری استفاده بکنیم که در مواقعی که کارت گرافیکی ما ویژگی را پشتیبانی نمی کند برای شبیه سازی آن ویژگی استفاده می شود.

۳- **RenderWindow**: مشخص می کند پنجره ای را برای **device** که ما در اینجا با استفاده از **this** فرم خود را مشخص کردیم.

۴- **Behavior flag**: برای کنترل جنبه های از **device** به کار می رود و می تواند چندین **flag** با هم ترکیب شوند تا چندین ویژگی را به یک **device** اعمال کنند.

۵- **Device Presents**: کنترل می کند نحوه نمایش داده ها بر روی صفحه را. حال ما باید تابعی که برای آماده سازی **device** نوشته ایم را در محل مناسب فراخوانی کنیم لذا کد زیر را در فایل **Program.cs** مینویسیم.

Static void Main ()

```
{
    Using (Form1 frm = new Form1 ())
    {
        // Show our form and initialize our graphics engine
        frm.Show ();
        frm.InitializeGraphics ();
        Application.Run (frm);
    }
}
```

کار بعدی که ما باید انجام دهیم قبل از شروع کار اصلی پاک کردن صفحه در مرحله اول بار گذاری است برای این کار ما از تابع **Clear** مربوط به شی **device** استفاده می کنیم.

```
device.Clear (ClearFlags.Target, System.Drawing.Color.CornflowerBlue, 1.0f, 0);
device.Present ();
```

برای نمایش تغییرات باید از متد **Present** استفاده کنیم.

اساسی ترین شی که ما در رسم های سه بعدی داریم رسم یک مثلث است ما با استفاده از مثلث های کافی می توانیم اشکال پیچیده را رسم کنیم لذا ضروریترین رسم مورد نیاز ما رسم مثلث است برای رسم مثلث ما نیاز به ۲

چیز داریم اول ساختمان داده که اطلاعات مربوط به مثلث را در خود نگه دارد دوم رسم واقعی مثلث توسط `device`.

ساختمان داده‌های مختلفی از کلاس `CustomVertex` برای نگه داری اطلاعات مثلث موجود است که مهمترین آنها ساختمان داده `PositionColored` که می‌تواند اطلاعات مربوط به راس مثلث در فضای سه بعدی را دارد و ما با استفاده از این ساختمان داده برای رسم یک مثلث واقعاً سه بعدی استفاده کنیم.

```
CustomVertex.PositionColored[] verts =
    new CustomVertex.PositionColored [3];
verts [0]. Position=new Vector3 (0.0f, 1.0f, 1.0f);
verts [0].Color = System.Drawing.Color.Aqua.ToArgb ();
verts [1]. Position =new Vector3 (-1.0f, -1.0f, 1.0f);
verts [1].Color = System.Drawing.Color.Black.ToArgb ();
verts [2]. Position =new Vector3 (1.0f, -1.0f, 1.0f);
verts [2].Color = System.Drawing.Color.Purple.ToArgb ();
```

منبع : [Tom Miller](#) By **Managed DirectX® 9 Kick Start: Graphics and Game Programming**

حالا ما با استفاده از کدهای بالا یک مثلث در فضای سه بعدی تعریف کردیم حال برای نمایش آن در صفحه نمایش باید کارهای زیر را انجام دهیم.

```
device.BeginScene ();
device.VertexFormat = CustomVertex.PositionColored.Format;
device.DrawUserPrimitives (PrimitiveType.TriangleList, 1, verts);
device.EndScene ();
```

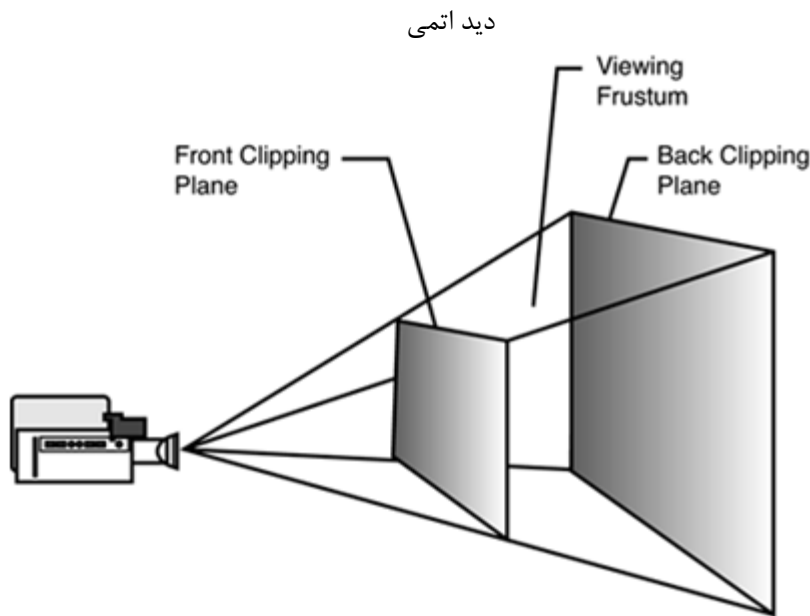
ما توسط متد `BeginScene` به `device` اجازه می‌دهیم اطلاعاتی در مورد رسم ما بدست بیاورد و آنرا آماده رسم کند..

توسط متد `DrawUserPrimitives` که رسم را به صورت واقعی انجام میدهد..
متد `EndScene` برای آگاهی `device` از اتمام رسم‌ها می‌باشد..

در مختصات سه بعدی ما دارای یک فضا هستیم و هیچ مبدا مختصاتی نداریم پس چطور این شکل را در صفحه نمایش دهیم؟ ما نیاز داریم به تعریف یک پرسپکتیو و یک دوربین که در واقع دید ما از فضای سه بعدی است برای این کار با توجه به شکل زیر و با استفاده از تابع `LookAtLH` و `PerspectiveFovLH` از کلاس `Matrix` می‌توانیم برای صحنه خودمان یک پرسپکتیوو یک دوربین تعریف کنیم و از دستورات زیر استفاده می‌کنیم.

```
Microsoft.DirectX.Matrix.PerspectiveFovLH (fieldOfViewY,
aspectRatio ,znearPlane ,zfarPlane )
```

```
Microsoft.DirectX.Matrix.LookAtLH(CameraPosition,
CameraTarget ,CameraUpVector )
```



شکل ۲-۱

```
Private void SetupCamera ()
{
    device.Transform.Projection = Matrix.PerspectiveFovLH ((float) Math.PI
    / 4, this.Width / this.Height, 1.0f, 100.0f);

    device.Transform.View = Matrix.LookAtLH (new Vector3 (0, 0, 5.0f),
    new Vector3(), new Vector3(0,1,0));
}

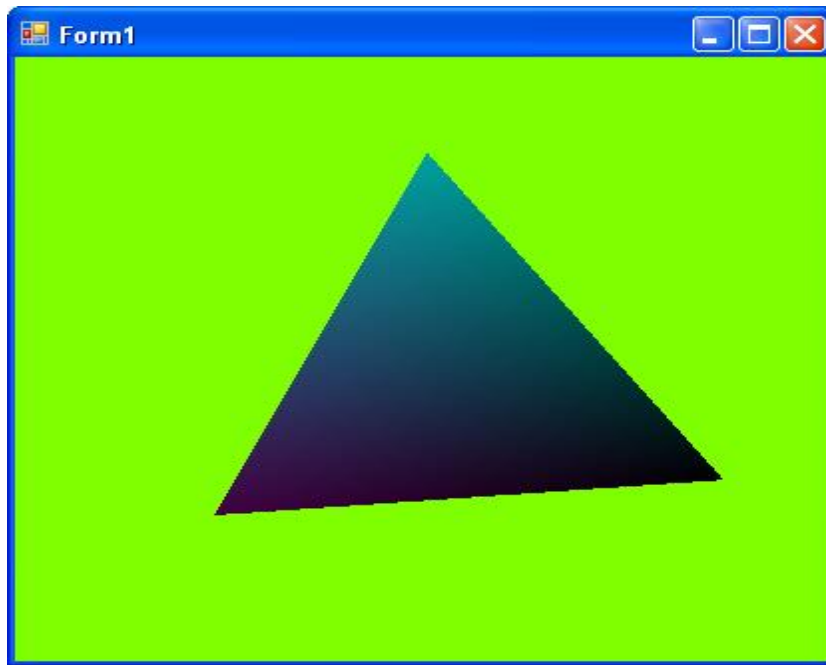
```

منبع : [Tom Miller](#) Managed DirectX® 9 Kick Start: Graphics and Game Programming

دستور اول با استفاده از قاعده دست چپ برای اعمال پرسپکتیو است و دستور دوم برای تعریف یک دوربین است که ما از طریق آن می توانیم محیط را ببینیم.

حال که ما تابع `SetupCamera` را تعریف کردیم باید انرا درون تابع `OnPaint` مربوط به `Form` اصلی فراخوانی کنیم تا پرسپکتیو به صحنه ی ما اعمال شود و ما بتوانیم اشیا را با اندازه مناسب به وسیله دوربین ببینیم. ولی زمانی که ما برنامه را اجرا میکنیم مثلث ما به صورت سیاه نمایش داده می شود این به خاطر این است که در حالت پیش فرض ما علاوه بر پرسپکتیو و دوربین باید برای محیط خود یک منبع نور تعریف کنیم تا بتوانیم اشیا درون محیط را با رنگ واقعی آنها مشاهده کنیم ما در اینجا فعلاً خاصیت نور پردازی برای محیط را خاموش میکنیم تا بتوانیم اشکال خود را با رنگ خودشان مشاهده نماییم تا بعد در بخش مربوط به نور پردازی یک منبع نور مناسب برای محیط خود تعریف کنیم. لذا با دستور زیر خاصیت `Lighting` مربوط به `Device` را `False` می کنیم. با اجرای برنامه خروجی زیر را در شکل ۳-۱ مشاهده خواهیم کرد.

```
device.RenderState.Lighting = false;
```

شکل ۱-۳

دوران:

در بخش دوران گفتیم که برای آنکه یک شکل را دوران بدهیم باید تمام مختصات راس های آن شکل را در ماتریس دوران ضرب کنیم تا آن شکل به صورت کامل حول نقطه مورد نظر به اندازه دلخواه دوران داده شود. همچنین ما میتوانیم چندین ماتریس انتقال دیگر مانند دوران و جابجایی را باهم ترکیب کنیم که DirectX این امکان را به وجود آورده است که ما با استفاده از کلاس Matrix و توابع استاتیک موجود در این کلاس تبدیلات لازم را به شی device خودمان اعمال کنیم که این تبدیلات در خصوصیت مربوط به word که یک ماتریس ۴*۴ است قرار میگیرد. کلاس Matrix دارای متدهای برای دوران حول محور X,Y,Z می باشد همچنین با استفاده از متد RotationAxis می تواند همزمان حول سه محور دوران کند. در کد زیر ما device را با استفاده از متد RotationZ حول محور Z به اندازه زاویه دورانی که بر حسب رادیان است دوران می دهیم.

```
device.Transform.World = Matrix.RotationZ((float)Math.PI / 6.0f);
```

نور پردازی:

از قبل به خاطر داریم که زمانی که ما خصوصیت مربوط به نور پردازی device را فعال میگردیم پشت اشکال تیره می شد یا باید یک منبع نور تعریف میگردیم یا باید این خصوصیت را غیر فعال می کردیم. ما می توانیم با تعریف کردن منبع نور مناسب جلوه های ویژه ای به اشکال خود بدهیم و تصاویر واقعی تری را به وجود بیاوریم لذا در اینجا به ایجاد یک منبع نور در برنامه می پردازیم.

انواع منبع نور:

- ۱- Point: این نوع منبع نور نور را در تمام جهات پخش می کند مانند یک لامپ.
- ۲- Directional: این منبع نور نور را در یک جهت ثابت و مستقیم پخش می کند مانند حورشید.
- ۳- Spot: این منبع نور نور را در یک جهت و در یک شعاع محدود پخش می کند مانند نوری که در سن پخش می شود.

کلاس DEVICE یک ارایه lights دارد با تعریف هر یک از اعضای ارایه خصوصیات گوناگونی را برای lights نگه داری می کنیم. لذا برای اعمال نور پردازی ابتدا ما باید خصوصیت Lighting مربوط به

device را true کنیم البته این خصوصیت به صورت پیش فرض True است. در مرحله بعد ما باید خصوصیات مربوط به هر عضو ارایه lights که میخواهیم تعریف کنیم را باید ست کنیم لذا اولین منبع نوری که تنظیم می کنیم میتواند به صورت زیر باشد.

```
device.Lights[0].Type = LightType.Point;
device.Lights[0].Position = new Vector3();
device.Lights[0].Diffuse = System.Drawing.Color.White;
device.Lights[0].Attenuation0 = 1.1f;
device.Lights[0].Range = 10.0f;
device.Lights[0].Enabled = true;
```

diffuse: رنگ مربوط به منبع نور ما می باشد.

Attenuation0: ضریب فرسایش نور می باشد.

Range: شعاعی که منبع نور در آن فعال است.

Position: برای تنظیم محل منبع نور می باشد که به صورت پیش فرض نقطه (0 و 0 و 0) در نظر گرفته می شود. با انجام تمام این کارها باز هم پشت شکل ما سیاه است چون محاسبات نور پردازی تنها با بردار نرمال انجام می گیرد.

بردار نرمال: برداری است که بر صفحه عمود است.

برای ایجاد بردار نرمال ما باید ساختمان دادهای که استفاده کردیم برای نگه داری راسها را عوض کنیم و ساختمان دادهای استفاده کنیم که علاوه بر مختصات راسها بتواند بردار نرمال را نیز نگه داری کند. لذا باید از ساختمان داده PositionNormalColored به جای PositionNormal استفاده کنیم پس تغییرات زیر را در برنامه خود به وجود می آوریم.

```
CustomVertex.PositionNormalColored[] verts = new
CustomVertex.PositionNormalColored[3];
verts[0].Position = new Vector3(0.0f, 1.0f, 1.0f);
verts[0].Normal = new Vector3(0.0f, 0.0f, -1.0f);
verts[0].Color = System.Drawing.Color.Aqua.ToArgb();
device.VertexFormat = CustomVertex.PositionNormalColored.Format;
```

منبع: [Tom Miller](#): Managed DirectX® 9 Kick Start: Graphics and Game Programming

استفاده از Vertex Buffers:

یک حافظه برای نگه داری راس ها میباشد ما با استفاده از Vertex Buffers می توانیم اشکال خود را روی صفحه تغییر دهیم .

برای ایجاد یک **Vertex Buffers** ما باید از کلاس **VertexBuffer** استفاده کنیم و یک شی از این کلاس به وجود آوریم.

```
Private VertexBuffer vb = null;
Vb = new VertexBuffer (typeof (CustomVertex.PositionColored), 3, device,
Usage.Dynamic | Usage.WriteOnly, CustomVertex.PositionColored.Format,
Pool.Default);
```

```
vb.SetData (verts, 0, LockFlags.None);
```

پارامترهای که به سازند **Vertex Buffers** ارسال می شوند عبارتند از :
device:Device ای که روی آن **Vertex Buffers** را ایجاد کردیم و تنها در این **Vertex device**
Buffers مورد قبول است.

SizeOfbufferInBytes: اندازه بافر به بایت که اگر ما از این مورد در سازنده استفاده کنیم **buffer** ما قادر خواهد بود انواع راسها را نگه دارد.

Numverts: ماکزیمم بافر قادر به نگه داری از آن است.

Usage: نحوه استفاده از بافر را معین میکند.

برای این که رسمهای ما به درستی عمل نمایند بای کهای زیر را جایگزین کدهای مربوط به رسم کنیم لذا داریم:
device.SetStreamSource(0, vb, 0);
device.DrawPrimitives(PrimitiveType.TriangleList, 0, 1);

ولی در اینجا یک مشکل وجود دارد و آن این است که زمانی که ما فرم خود را رفرش میکنیم دیگر در صفحه چیزی نمایش داده نمیشود و به این علت است هر زمان که رویداد **Onpaint** مربوط به بافر از بین میرود و بافر مجدد ایجاد میشود و ما باید در زمان ایجاد دادهای خودمان را مجدد در بافر بار کنیم لذا نیاز داریم رویداد **Create** بافر را سربارگذاری کنیم لذا داریم

```
private void OnVertexBufferCreate(object sender, EventArgs e)
{
    VertexBuffer buffer = (VertexBuffer)sender;
    CustomVertex.PositionColored[] verts = new CustomVertex.PositionColored[36];
    verts[0] = new CustomVertex.PositionColored(-1.0f, 1.0f, 1.0f, Color.Red.ToArgb());
    verts[1] = new CustomVertex.PositionColored(-1.0f, -1.0f, 1.0f, Color.Red.ToArgb());
    verts[2] = new CustomVertex.PositionColored(1.0f, 1.0f, 1.0f, Color.Red.ToArgb());
    verts[3] = new CustomVertex.PositionColored(-1.0f, -1.0f, 1.0f, Color.Red.ToArgb());
    verts[4] = new CustomVertex.PositionColored(1.0f, -1.0f, 1.0f, Color.Red.ToArgb());
    buffer.SetData(verts, 0, LockFlags.None);
}
vb.Created += new EventHandler(this.OnVertexBufferCreate);
OnVertexBufferCreate(vb, null);
```

منبع : [Tom Miller](#) By Managed DirectX® 9 Kick Start: Graphics and Game Programming

بافتها:

زمانی که ما اشیا را با رنگ و نور نمایش می دهیم زیاد این نمایش واقعی به نظر نمی رسد لذا برای اینکه تصویر ما واقعی به نظر برسد از بافتها استفاده می کنیم.

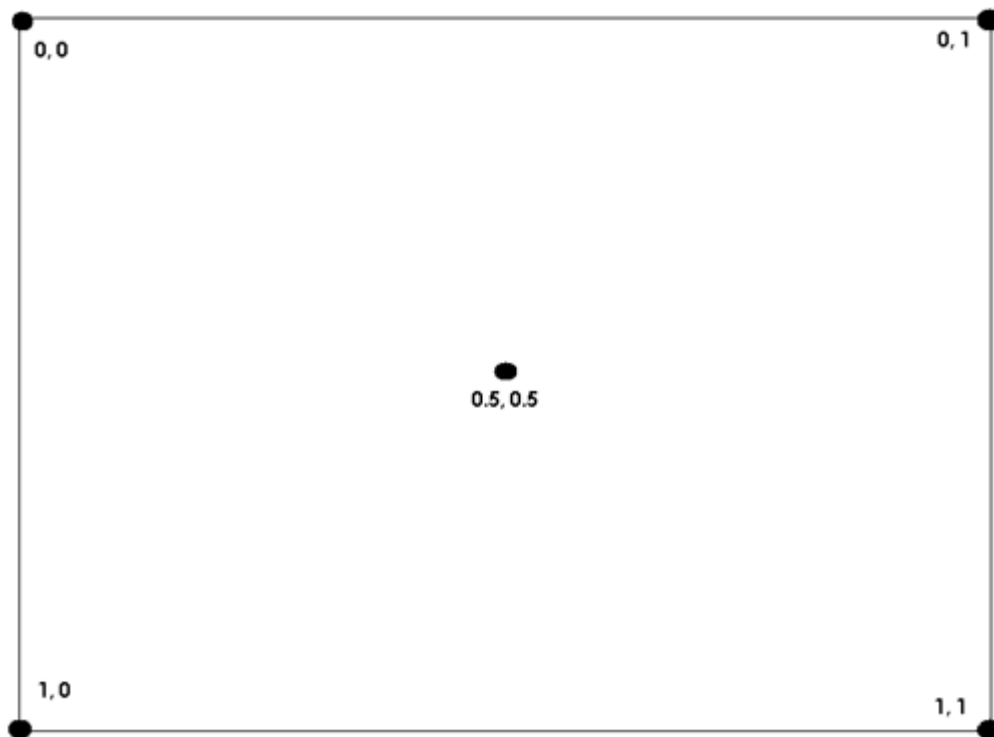
Textures: Textures در صحنه های ۳ بعدی در اصطلاح یک تصویر ۲ بعدی bitmap که استفاده می شود برای شبیه سازی بافتها روی اشیای اولیه مانند نقطه و مثلث .

Direct3D می تواند رندر کند تا هشت بافت را برای هر شکل پایه در یک زمان.

چگونه یک Bitmap2D مسطح تبدیل می شود به چیزی که می تواند روی یک شکل سه بعدی رسم شود؟

هر شی که ارائه میشود به صحنه نیاز به مختصات بافت دارد که استفاده می شود برای نگاشت هر نقطه از تصویر به یک پیکسل متناظر در صحنه در زمان تبدیل.

بر روی هر تصویر ما پنج نقطه را به عنوان مختصات UV در نظر می گیریم که این نقاط در شکل زی نمایش داده شده اند. که این نقاط گوشه بالا سمت راست و چپ و گوشه پایین سمت راست و پایین و مرکز تصویر است.



شکل ۱-۴

لذا ما باید با استفاده از این نقاط یک برش از تصویر را روی شکل اولیه مانند مثلث نگاشت کنیم که برای این کار لازم داریم که راس هایی که تعریف می کنیم از نوع ساختمان داده PositionTextured باشد که در آن باید مختصات UV را هم باید اعمال کنیم نمونه آن در مثال زیر آمده است.

```
CustomVertex.PositionTextured[] verts = new
```

```
CustomVertex.PositionTextured[18];
```

```
verts[0] = new CustomVertex.PositionTextured(-1.0f, 0.5f, 1.0f, 0.0f, 0.0f);
verts[1] = new CustomVertex.PositionTextured(-1.0f, 0.5f, -1.0f, 0.0f, 1.0f);
verts[2] = new CustomVertex.PositionTextured(1.0f, 0.5f, -1.0f, 1.0f, 1.0f);
```

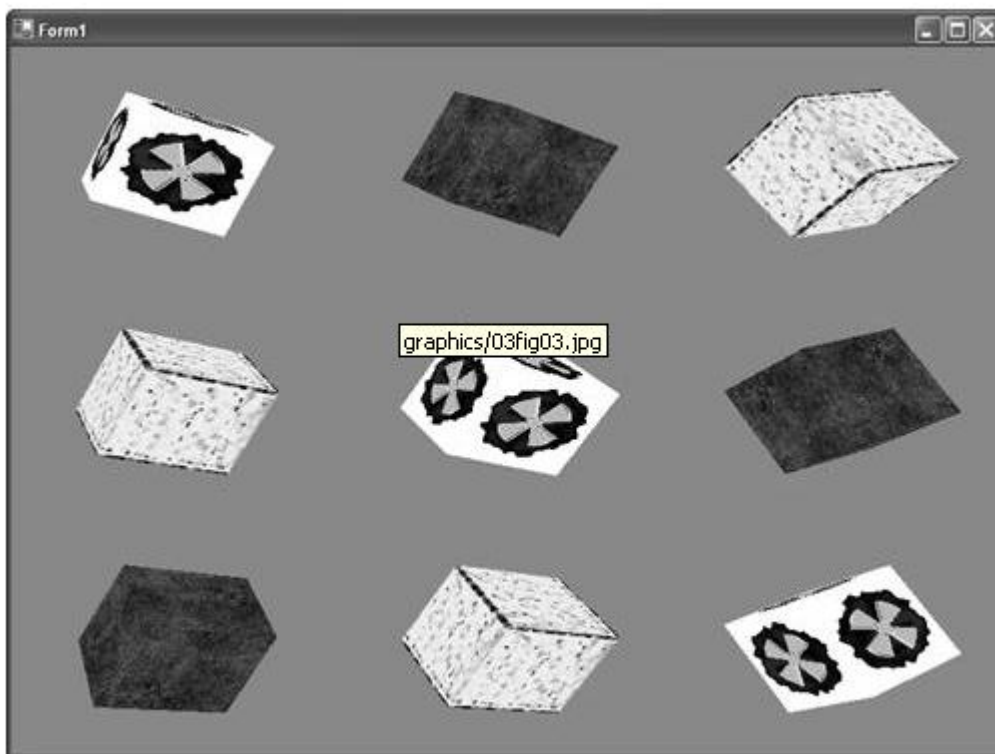
در گام بعدی ما لازم داریم که یک شی از نوع Texture را تعریف کنیم :

```
private Texture text = null;
```

سپس ما باید این شی را با یک Bitmap ست کنیم که توسط کد زیر این کار را انجام می دهیم.

```
text = new Texture(device, new Bitmap(this.GetType(),
"River.bmp"), 0, Pool.Managed);
device.SetTexture(0, tex);
```

با نوشتن این کد در واقعه ما بافت خود را به وجود آوردیم و آن را به Device اعمال کردیم. ما می توانیم بافت‌های زیادی را به Device اعمال کنیم و اشکال مختلف با بافت‌های مختلف به وجود آوریم.



شکل ۱-۵

:Mesh

معمولا می‌خواهیم از اطلاعات مربوط به راس‌ها را از یک منبع خارجی مثل فایل لود کنیم. مانند اطلاعات مربوط به رسم یک مکعب یا کره یا شی سه بعدی پیچیده‌ای که شامل صدها راس می باشد که در DirectX این اطلاعات درون یک فایل با فرمت X ذخیره می شود.

در DirectX وجود دارد یک شی که کپسوله می کند اطلاعات راس ها را که نامیده میشود Mesh.

Mesh: نام دیگری برای اجسام سه بعدی ، مش ها از اجسام اولیه ای مثل خط و مثلث تشکیل شده اند و میتوانند دارای بافت بوده تحت نور پردازی هم قرار بگیرند.

مش ها می توانند نگه داری بکنند انواع داده های گرافیکی ولی معمولا برای کپسوله کردن مدل های پیچیده به کار می رود. مش ها دارای توابعی برای بالا بردن سرعت رندر کردن اشیا. تمام مش ها دارای یک **Index Buffer** و یک **VertexBuffer** هستند.

ما برای شروع کار با مش ها نیاز داریم ابتدا اسمبلی `Microsoft.DirectX.Direct3DX.dll` را به رفرنس های پروژه خود اضافه کنیم سپس می توانیم یک شی از نوع **Mesh** ایجاد نماییم.

```
private Mesh mesh = null;
```

وجود دارد چندین متد استاتیک که ما میتوانیم از آنها استفاده کنیم برای ایجاد کردن و لود کردن آنها استفاده کنیم مانند متد `Box` که همانطور که از اسمش پیدا است برای رسم یک مکعب استفاده می شود که پارامترهای آن شامل `Device` مورد نظر طول و ارتفاع و عرض است.

```
mesh = Mesh.Box(device, 2, 2, 2);
```

هر مش شکسته می شود به زیر مجموعه هایی که با استفاده از تابع `DrawSubset` ما هر یک از این زیر مجموعه ها را رسم می کنیم که در مش های ساده این مجموعه یکی است.

```
mesh.DrawSubset(0);
```

چون در مش ها فقط مختصات و بردار نرمال نگه داری می شود و رنگ نگه داری نمی شود لذا زمانی که نور پردازی خاموش است تمام چیزها سفید است و وقتی نور پردازی روشن است تمام چیزها سیاه است چون منبع نوری تعریف نکردیم ما برای اعمال رنگ می توانیم از نور محیطی استفاده کنیم.

نور محیطی: نوعی نور که بعثت پخش زیاد ، روی همه نقاط صحنه اثر یکسانی دارد و در واقع برای تنظیم

روشنایی کلی محیط بکار میرود . ماده ها نیز میتوانند بر حسب خصوصیت محیطی خود به این نور پاسخ دهند .

```
device.RenderState.Ambient = Color.Blue;
```

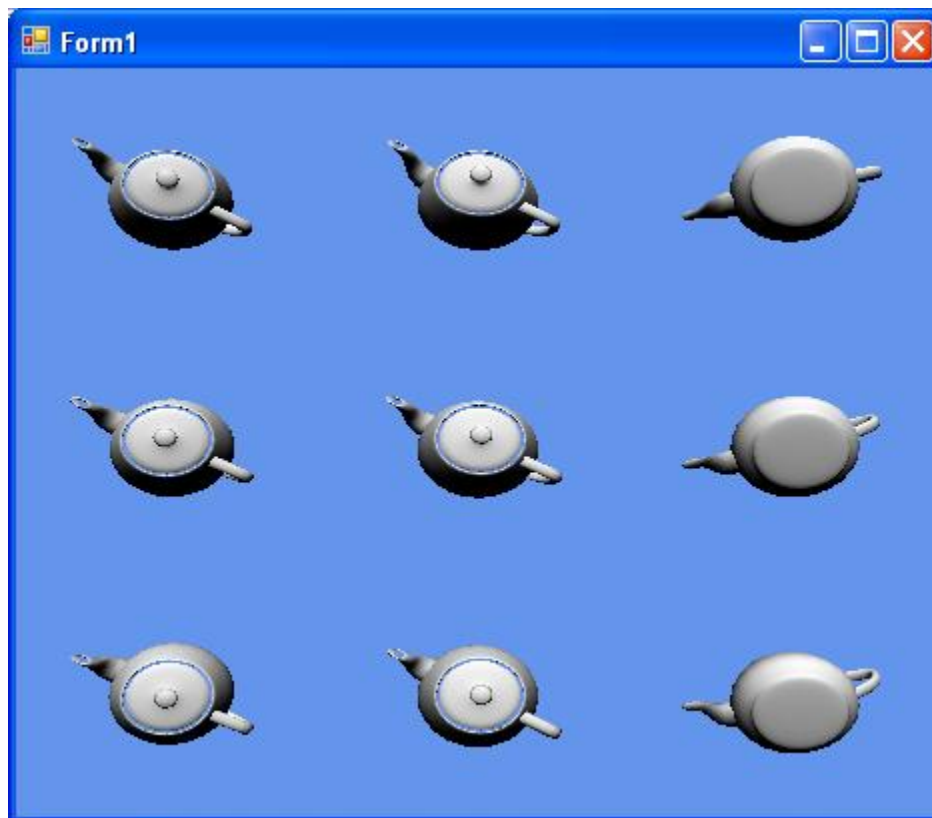
نور محیطی روی تمام صحنه تاثیر دارد و باعث می شود تمام اشکال رنگ بگیرند که در اینجا اشکال روی صحنه رنگ ابی می گیرند.

استفاده از ماتریال ها و نور پردازی:

در `Direct3D` برای اینکه به درستی محاسبه کند رنگ یک نقطه خاص را در اشیا سه بعدی علاوه بر رنگ آن نقطه باید توصیف کنیم بازتاب نور توسط سطح اشیا را که این کار توسط ماتریال ها انجام می شود.

```
boxMaterial = new Material();
boxMaterial.Ambient = Color.Silver;
boxMaterial.Diffuse = Color.White;
device.Material = boxMaterial;
```

در واقعه با نسبت دادن یک ماتریال به صحنه ما بازتابش صحیح نور را به وجود آوردیم ولی برای داشتن اشکال واقعی تر علاوه به ماتریال به منبع نور مناسب که قبلا گفته شد نیاز داریم. در شکل زیر تاثیر ماتریال را بر بازتاب صحیح نور مشاهده می کنید.



انواع دیگر مش:

۱-سیلندر:

```
mesh = Mesh.Cylinder(device, Radius1, Radius2, Length, Slices, Stacks)
```

۲-چند ضلعی:

```
mesh = Mesh.Polygon(device, Length, Sides);
```

۳-قوری:

```
mesh = Mesh.Teapot(device);
```

۴-کره:

```
mesh = Mesh.Sphere(device, Radius1, Slices, Stacks);
```

۵-حلقه:

```
mesh = Mesh.Torus(device, InnerRadius, OuterRadius, Sides, Rings);
```

slices : تعداد برش ها حول محور اصلی.

Stacks : تعداد پشته حول محور اصلی.

منابع:

Managed DirectX® 9 Kick Start: Graphics and Game Programming By [Tom Miller](#)