مقاله آموزشی برنامه نویسی Direct3D توسط C#

آشنایی با ریاضیات D3D و مراحل Rendering Pipeline

طراحی و تنظیم:میثاق
www.Persian-Designers.com
Game_Programming2002@yahoo.com

بعد از بارگذاری نوبت به رسم گرافیک توسط DX میرسه.اما قبل از شروع دستورات ترسیمی خوبه تا کمی با محاسبات سه بعدی در Direct3D آشنا بشیم.

بردارها:

تعریف عمومی یک بردار، پاره خط جهت داریه که دارای دوخصوصیت اصلی یعنی اندازه و جهت است. در D3D از بردارها به منظور تعیین جهت شعاعهای نوری،تعیین جهت قرارگیری اجسام و نیز تعیین جهت دید دوربینها استفاده میشه.
بردارها در دوحالت کلی قابل ترسیم هستند،حالت اول زمانیه که مختصات نقاط ابتدا و انتهای اونا دلخواه باشه در اینصورت به بردار بردار آزاد گفته میشه.
حالت دوم زمانیه که ابتدای بردار در مرکز مختصات باشه و با دونستن فقط نقطه انتهایی بردار رسم بشه.در این حالت به بردار بردار استاندارد گفته میشه. در ریاضیات D3D بردارها همگی در حالت استاندارد رسم میشن.
اما پیش از هر رسمی باید بدونیم دستگاه مختصات در D3D چه حالتی داره.
در ریاضیات سه بعدی دو دستگاه مختلف معروف وجود داره،اولی دستگاه مختصات دست چپه که در اون جهت Y مثبت به سمت بالا،X مثبت به سمت راست و Z مثبت به سمت داخل صفحه است.
دومی دستگاه مختصات دست راسته که تفاوتش با دست چپ در جهت محور Z مثبته که به سمت خارج صفحه است.
باید بدونید که دستگاه مختصات D3D دستگاه دست چپه.
خب برگردیم سر بردارها در D3D برای نگهداری اطلاعات یک بردار از کلاس Vector3 استفاده میشه.به کد زیر توجه کنید:

```
Vector3 vec3=new Vector3(x,y,z);
```

در این کد ابتدا متغیر vec3 تعریف شده سپس همون جا تبدیل به شئ Vector3 میشه و مقدار میگیره.در این حالت مختصات x,y,z انتهای بردار در vec3 قرار داده میشه. البته می تونیم شئ vec3 رو بعدا مانند زیر مقدار بدیم:

```
Vector3 vec3=new Vector3();
vec3.X=x;
vec3.Y=y;
vec3.Z=z;
```

البته در کلاس Vector3 توابع و عملگرهای مفیدی برای کار با بردارها تعبیه شده که اونها رو بررسی می کنیم.

عملگرهای "==" و "=!":

از این دو عملگر برای مقایسه دو بردار استفاده میشه به کد زیر توجه کنید:

```
Vector3 vec3_1=new Vector3(1.0f,2.0f,3.0f);
Vector3 vec3_2=new Vector3(2.0f,4.0f,0.0f);
if(vec3_1==vec3_2).....
```

تابع Length:
از این تابع برای بدست آوردن اندازه بردار استفاده میشه.فرمول بدست آوردن اندازه یک بردار عبارت است از:جذر x^2+y^2+z^2 .
که این تابع برای ما محاسبه میکنه.

```
float len;
len=vec3.Length;
```

تابع Normalize :
از این تابع برای نرمالسازی یک بردار استفاده میشه.در نرمالسازی یک بردار تمام مولفه های اون بردار رو بر اندازه بردار تقسیم می کنیم تا مولفه های بردار نرمال بدست بیاد.

```
vec3.Normalize();
```

این دستور نرمال بردار vec3 رو داخل خودش قرار می ده.
عملگرهای + و -:
این دو عملگر عمل جمع و تفریق دو بردار رو انجام داده و حاصل رو در بردار دیگه ای قرار می دن.

```
vec3_Result=vec3_1+vec3_2;
vec3_Result=vec3_1-vec3_2;
```

در بالا vec3_Result,vec3_1,vec3_2 همگی از نوع Vector3 هستند.
عملگر *:
از این عملگر برای ضرب یک عد در بردار استفاده میشه.و عدد وارد شده در تک تک مولفه های بردار ضرب میشه.

```
vec3_Result=vec3_1*5.0f;
```

تابع Dot :
از این تابع در ضرب نقطه ای دو بردار استفاده میشه.روش ضرب نقطه ای دو بردار بدین صورته که،حاصلضرب مولفه های x دو بردار بعلاوه حاصلضرب مولفه های Y دو بردار بعلاوه حاصلضرب مولفه های Z دو بردار . حاصل این عملیات یک عدد است و استفاده های زیادی از این عدد میشه.

```
float x;
x=Vector3.Dot(vec3_1,vec3_2);
```

اگر x مساوی با صفر باشه یعنی دو بردار vec3_1,vec3_2 بر هم عمودند.
اگر x بزرگتر از صفر باشه ،زاویه بین دو بردار کوچکتر از ۹۰ درجه است و
اگر x کوچکتر از صفر باشه،زاویه بین دو بردار بزرگتر از ۹۰ درجه است.
تابع Cross:
از این تابع برای بدست آوردن بردار حاصلضرب بین دو بردار استفاده میشه. فرمول ریاضی این ضرب بصورت زیره:
اگر x1,y1,z1 مولفه های بردار vec3_1 و x2,y2,z2 مولفه های بردار vec3_2 باشن مولفه های بردار حاصل یعنی vec3_Result عبارتند از:

```
vec3_Result=[(y1*z2-z1*y2),(z1*x2-x1*z2),(x1*y2-y2*x1)]
```

و تابع Cross بصورت زیر نوشته میشه:

```
vec3_Result=Vector3.Cross(vec3_1,vec3_2);
```

نکته مهم در این ضرب اینه که بردار حاصلضرب یعنی vec3_Result بر هر دوبردار vec3_1 و vec3_2 عموده.
ماتریس:
خب بعد از بررسی بردارها نوبت به بررسی یکی دیگه از اجزای مهم در ریاضیات سه

بعدی و همچنین در D3D میشه که عبارت است از ماتریس ها.
ماتریس آرایه ای از اعداده که در یک یا چند سطر و ستون مرتب شدن.و از اون برای نگهداری اطلاعات مرتبط با هم استفاده میشه. برای نشون دادن یک ماتریس از نام ماتریس بهمراه تعداد سطر و ستون در اون ماتریس استفاده میشه.به عنوان مثال در زیر ماتریس U معرفی شده که ۳ سطر و ۵ ستون داره:

U3*5

در مورد ماتریس هایی که فقط یک سطر یا ستون دارن به ترتیب اونها رو ماتریس سطری و ماتریس ستونی می گن.
در D3D از ماتریسها برای انتقالات استفاده میشه. مثلا اگر بخوایم یه شئ رو تحت زاویه خاصی بچرخونیم،یا شئ رو به مکان مورد نظر انتقال بدیم و... ابتدا مشخصات انتقال رو در یک ماتریس ذخیره کرده و سپس اون ماتریس رو به تابع مورد نظر می دیم تا عمل انتقال انجام بشه.
در D3D برای نگهداری اطلاعات یک ماتریس از کلاسی به نام Matrix استفاده میشه.

```
Matrix m=new Matrix();
```

این حالت کلی تعریف یک ماتریسه ، البته ما معمولا ماتریس رو مانند روش بالا به کار نمی بریم.بلکه کلیه اعمال انتقال توسط D3D محاسبه میشه و در ماتریس مربوطه قرار داده میشه و ما فقط ماتریس آماده شده رو برای انتقال به یه تابع منتقل می کنیم.
اما بهرحال در مواقعی لازمه تا ماتریسهای دلخواهمون رو تعریف کنیم.در اینصورت از کد بالا استفاده میشه.

تساوی دو ماتریس:

دو ماتریس زمانی با هم برابرند که درایه های نظیر به نظیر اونها با هم برابر باشن.البته در D3D با استفاده ار عملگر"==" در کلاس Matrix می تونیم برابری دو ماتریس رو بررسی کنیم.

توابع Add و Subtract :

از این دو تابع به منظور جمع و تفریق کردن دو ماتریس استفاده میشه. در جمع دو ماتریس درایه های نظیر به نظیر دو ماتریس با هم جمع شده و درایه های ماتریس حاصل جمع رو می سازن.در تفریق هم درست مثل جمع عمل میشه.توجه داشته باشین که در جمع و تفریق دو ماتریس باید از نظر سطر و ستون با هم برابر باشن.

```
Matrix m_Result=new Matrix();
Matrix m_1=new Matrix();
Matrix m_2=new Matrix();
m_1(0,0,1,....);
m_2(1,1,2,....);
m_Result=Matrix.Add(m_1,m_2);
m_Result=Matrix.Subtract(m_1,m_2);
```

تابع Multiply:

این تابع عمل ضرب دو ماتریس رو انجام می ده.توجه داشته باشین که در ضرب تعداد سطرهای ماتریس اول باید با تعداد ستونهای ماتریس دوم برابر باشه در غیر اینصورت ضرب ماتریسها عملی نیست.

```
m_Result=Matrix.Multiply(m_1,m_2);
```

متد Identity:

این متد یک ماتریس Identity چهار در چهار رو برای ما ایجاد میکنه.ماتریس Identity ماتریسیه که تمام درایه های روی قطر اصلیش برابر یک و مابقی درایه هاش برابر صفره.و از ضرب هر ماتریس در ماتریس Identity خود اون ماتریس بدست میاد. یعنی

ماتریس Identity حکم عدد یک رو در ضرب داره.کد زیر یک Identity چهار در چهار رو در ماتریس m_Result قرار می ده.

```
m_Result=Matrix.Identity;
```

تابع TransposeMatrix:

این تابع ماتریس Transpose مربوط به ماتریسی رو که به عنوان آرگومان ورودی بهش میدیم بر می گردونه. Transpose یک ماتریس از عوض کردن جای سطرها و ستونهای اون ماتریس بدست میاد.مثلا اگر ما یک ماتریس ۳*۲ (۲ تا سطر و ۳ تا ستون) داشته باشیم ، Transpose این ماتریس یک ماتریس ۲*۳ میشه.

```
m_Result=Matrix.TransposeMatrix(m_1);
```

البته عملگرها و توابع زیادی در کلاس ماتریس وجود داره که قادره اعمال مختلفی رو روی ماتریسها انجام بده.اما چون بحث اصلی ما نحوه تعریف انتقاله از اونا صرف نظر می کنم و بعضی از اونها رو در جاهایی که لازمه بطور کامل توضیح می دم.اگر مایل به کسب اطلاعات بیشتر هستید سوالات خودتون رو در تاپیک DirectX مطرح کنید تا دیگران هم از اون استفاده کنن.

انتقال:

انتقال عبارت است از هرگونه جابجایی در مکان ، زاویه ، و یا ابعاد یک شئ.
برای تعریف یک جابه جایی از یک ماتریس ۴*۴ استفاده میشه.سپس این ماتریس به تابع SetTransform در شئ device انتقال داده میشه تا بر اساس این ماتریس انتقال مورد نظر انجام بشه.البته به هیچ وجه نیاز نیست که ما بدونیم D3D محاسبات مربوط به این انتقالها رو چه طوری انجام می ده و چطوری از ماتریسهای ۴*۴ باری نگهداری این انتقالها استفاده میکنه.اما بازهم اگر تمایل به کسب اطلاعات بیشتر داشتین طبق معمول به تاپیک DirectX مراجعه کنید.
در زیر برخی از توابع مهم در کلاس Matrix که عملیات انتقال رو برای ما انجام می دن اشاره کردم.

تابع Translation:

از این تابع برای انتقال یک شئ به مختصات دلخواهمون در فضای 3D استفاده می کنیم.برای استفاده از این تابع ابتدا یک ماتریس تعریف کرده سپس اون رو به عنوان خروجی تابع فوق قرار میدیم.(در اینجا ماتریس m_Result) سپس تابع رو بصورت زیر استفاده می کنیم:

```
m_Result=Matrix.Translation(x,y,z);
```

x,y,z مختصات محل مورد نظر ما در فضای 3D است. پس از اجرای این تابع ماتریس انتقال مورد نظر توسط D3D محاسبه میشه و داخل m_Result قرار داده میشه.سپس ما m_Result رو به عنوان ورودی تابه SetTransform به کار می بریم تا انتقال ما انجام بشه.بهمین سادگی!

تابع RotationX,RotationY,RotationZ:

از سه تابع فوق برای چرخوندن یک شکل حول یکی از محورهای x,y,z استفاده میشه.این توابع به عنوان آرگومان ورودی زاویه مورد نظر بر حسب رادیان رو دریافت میکنن سپس ماتریس انتقال مورد نظر ما رو می سازن.

```
m_Result=Matrix.RotationX(30);
```

تابع Scaling:

از این تابع برای تغییر اندازه شئ در راستای هریک از سه محور x,y,z استفاده میشه.
این تابع یه آرگومان x,y,z رو به عنوان ورودی میگیره و جسم رو در راستای محورها به اندازه عدد وارد شده تغییر ابعاد می ده.

```
m_Result=Matrix.Scaling(1f,2f,1f);
```

توابع مربوط به انتقال نقاط و بردارها:

توابعی رو که تا حالا بررسی کردیم، همه توانایی انتقال یک شئ 3D مثلا یک مکعب و یا انتقال قسمتی از فضای سه بعدی رو دارن.اما اگر خواسته باشیم یک نقطه یا یک بردار رو انتقال بدیم باید از توابع خاصی استفاده کنیم.

تابع TransformCoordinate:

این تابع یکی از توابع کلاس Vector3 می باشد.از این تابع برای انتقال یک نقطه استفاده میشه.ما نقطه رو توسط یک بردار استاندارد مشخص می کنیم.این تابع دو آرگومان رو به عنوان ورودی می گیره که اولی برداری از کلاس Vector3 است که نقطه رو مشخص می کنه و دومی ماتریس انتقال مورد نظر ماست.این تابع مختصات بردار جدید رو بر می گردونه.

```
Vector3 vec1,vec2;
Matrix.Transformation m1;
vec1(1,1,1);
vec2=Vector3.TransformCoordinate(vec1,m1);
```

تابع TransformNormal:

این تابع برای انتقال یک بردار مورد استفاده قرار میگیره و در کلاس Vector3 قرار داره تمامی آرگومانها و خصوصیات این تابع مثل تابع بالاست با این تفاوت که این تابع بردار رو به معنی واقعی خود بردار در نظر میگیره در حالیکه TransformCoordinate بردار رو وسیله ای برای تعریف نقطه فرض می کنه.

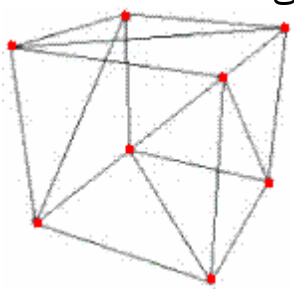
آمادگی برای ترسیم:

همونطور که می دونید شکل پایه در ترسیم تمامی اشکال سه بعدی مثلثه.یعنی از ساده ترین شکل سه بعدی یعنی مکعب گرفته تا پیچیده ترین مشها از کنار هم قرار گرفتن مثلثها بوجود می آید.

بنابراین اولین قدم در رسم آشنایی با نحوه ترسیم مثلثهاست. هر مثلث سه راس و سه ضلع داره که این سه راس رو بهم وصل میکنه. در D3D برای رسم مثلث کافیه تا مختصات رئوسش رو تعریف کنیم سپس D3D به طور خودکار رئوس رو طبق ترتیب مشخص شده بهم وصل میکنه. البته در تعریف رئوس امکانات بسیار متنوعی در اختیار ما قرار داره که در مقاله بعدی بطور کامل با اونها آشنا میشیم.

تا اینجا فقط بدونید برای رسم هر شکل سه بعدی به تعداد کم یا زیادی مثلث نیازه.

یه عنوان مثال برای رسم مکعب ما نیاز به ۱۲ مثلث داریم؛ که این به معنی تعریف ۳۶ راسه؛که هر چهار تا راس یک وجه مکعب رو تعریف می کنه.



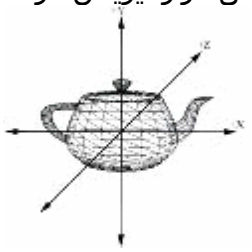
همنطور که در شکل میبینید از ۳۶ راس تشکیل دهنده فقط ۸ تای آنها با اضلاع غیر مشترک بهم مربوطند و مابقی اضلاع، بین این ۸ راس مشترکند.بنابراین برای بالا رفتن سرعت اجرای برنامه و نیز کم شدن حجم کد نویسی از تکنیک دیگری به نام Indexing استفاده میشه.در این حالت ما رئوس غیر مشترک رو تعریف کرده و رابطه بین این رئوس رو در IndexBuffer مشخص می کنیم.منظور از رابطه بین رئوس اضلاعی است یه از یک راس به رئوس دیگر مربوط می شوند.

Rendering Pipeline:

به عملیات مربوط به رسم اشکال تا نمایش آنها بر روی مانیتور RederingPipeline گفته میشه.این مراحل عبارتند از:

۱-رسم شکل در LocalSpace:

در این مرحله ما شکل مورد نظر خودمون رو در LocalSpace یا همون اطراف مرکز مختصات رسم می کنیم.علت این کار اینه که هم رسم و هم ویرایش اشکال در LocalSpace راحت تره. بعد از رسم شکل ما اون رو به محل قرارگیریش در دنیای سه بعدی منتقل می کنیم.



۲-انتقال شکل به WorldSpace:

در این مرحله توسط یک ماتریس انتقال Translation شکل رو به مختصات اصلیش منتقل می کنیم. نحوه ایجاد یک ماتریس انتقال توسط تابع Translation در کلاس Matrix رو یادداریدد فقط می مونه نحوه اعمال این انتقال یه شکل. برای این منظور از تابع SetTransform در کلاس device استفاده می کنیم. این تابع دو آرگومان ورودی دارد اولین آرگومان مشخص کننده نوع انتقاله.چون در اینجا ماتریس انتقال ما حاوی انتقال شکل به مختصات جدیدش در دنیا است ما آرگومان اوی رو World انتخاب می کنیم. و آرگومان دوم ماتریس انتقال ماست.

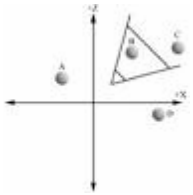
```
Matrix m_translation=new Matrix();
m_translation=Matrix.Translation(10f,20f,25f);
device.SetTransform(TransformType.World,m_translation);
```

۳-تعیین ViewSpace:

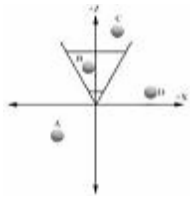
بعد از طراحی دنیای سه بعدیمون نوبت به ایجاد یک دوربین می رسه تا از طریق اون بتونیم به بخشهای مختلف دنیا نگاه کنیم.D3D برای راحت تر شدن محاسباتش تنها از یک دوربین استفاده میکنه که در مبدا مختصات قرار گرفته و در جهت محور z مثبت نگاه می کنه.اما اگر ما بخوایم دوربین رو در محل دلخواهمون قرار بدیم و به جهت دلخواهی نگاه کنیم باید چه کار کنیم؟

برای تعریف دوربین به سه مولفه نیاز داریم:۱-محل قرارگیری دوربین،۲-جهتی که دوربین به اون نگاه می کنه،۳-مختصات بالا در دنیا،که معمولا محور y مثبت انتخاب میشه.

در شکل زیر دوربین در مکانی دلخواه و با جهتی دلخواه قرار گرفته، در این میدان دید تنها شکل B قابل دیدن است.



اما چون دوربین D3D فقط می تواند از مبدا مختصات و در جهت z مثبت نگاه کند نیاز به یک انتقال احساس می شود. در این انتقال ما مختصات مورد نظر رو برای دوربین در قالب یک ماتریس در میاریم و سپس توسط SetTransform کلیه اشیا در دنیا توسط اون ماتریس انتقال پیدا می کنند تا میدان دید مورد نظر ما مقابل دوربین D3Dقرار بگیره.



در شکل بالا دنیای ما توسط یک ماتریس انتقال چرخش پیدا کرده تا همون میدان دید قبلی رو در مقابل دید دوربین قرار بده. برای تعریف ماتریس انتقال ViewSpace از تابع LookAtLH موجود در کلاس Matrix استفاده میشه.این تابع سه بردار رو به عنوان آرگومان ورودی میگیره. اولین بردار یک بردار استانداردە که برای مشخص کردن یک نقطه در دنیا که همون محل قرارگیری دوربین است به کار میره.بردار دوم برای مشخص کردن جهت نگاه کردن دوربین به کار میره.و سومین بردار جهت بالا رو در دنیا مشخص می کنه.

```
Vector3 position=new Vector3(5f,2f,-5f);
Vector3 target=new Vector3(0f,0f,0f);
Vector3 up=new Vector3(0f,1f,0f);
Matrix m_viewspace=new Matrix();
m_viewspace=Matrix.LookAtLH(position,target,up);
device.SetTransform(TransformType.View,m_viewspace);
```

در خط آخر کد فوق تابع SetTransform توسط پرچم تنظیمی View تنظیم شده.
این پرچم به تابع میگه ماتریس ورودی مشخصات یک انتقال ViewSpace رو در خودش داره،تا تابع انتقالات رو بطور صحیح انجام بده.

۴-عمل Backface Culling :

بعد از ترسیم اشکال و قرارگرفتن اونا مقتبل دوربین، اگر شئ جامد باشه فقط بخش جلوییش که رو به دوربینه دیده میشه و قسمت پشتی اون مخفی است.
D3D برای صرفه جویی در فضای حافظه کارت گرافیک و نیز سرعت اجرای برنامه بخش پشتی اشکال رو که دیده نمی شن حذف می کنه.به این عمل Backface Culling میگن.
نحوه تعریف D3D از جلو و پشت یک جسم به نحوه قرارگیری اضلاع مثلثهای تشکیل دهنده اون جسم بستگی داره.در حالت پیش فرض D3D سمتی رو که جهت اضلاع در جهت عقربه های ساعته جلوی جسم و سمت دیگه رو پشت جسم تلقی میکنه.البته با استفاده از تابع SetRenderState در کلاس device این حالت قابله تغییره.
تابع SetRenderState تابعیه که در هنگام رسم یک شکل،نحوه ترسیم رو مشخص میکنه. دراینجا ما از این تابع برای تغییر پیش فرض D3D در شناسایی جلو و پشت یک جسم ایتفاده می کنیم.

```
device.SetRenderState(RenderStates.CullMode,{1 or 2 or 3});
```

۱-BackFace Culling رو غیر فعال میکنه.
۲-قسمتهایی که جهت اضلاع در جهت عقربه های ساعته پشت تلقی میشه.
۳-پیش فرض D3D.

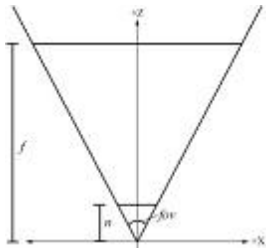
۵-Lighting:

مبحث مربوط به نور افشانی رو در جای خودش توضیح می دم.

۶-Clipping:

بعد از تعریف یک دوربین برای بازیابی بخشی از فضا D3D کلیه اشیائی رو که بطور کامل در میدان دید دوربین قرار داشته باشن بطور کامل نشون می ده.اما اشیائی رو که فقط قسمتی از اونها در میدان دید قرار داره ، همون بخش رو نشون میده و بخشی که فعلا در میدان دید نیست حذف میکنه.و سایر اشیائ رو که بطور کامل در میدان دید نیستند برای افزایش سرعت موقتا و بطور کامل از حافظه خارج میکنه.

این عمل رو که بطور خودکار انجام میگیره Clipping میگن.
۷-Projection:
بعد از مشخص کردن محل قرارگیری دوربین و جهت آن،نوبت می رسه به تعریف یک میدان دید معین. یکی از راههای بوجود آوردن فضاهای سه بعدی استفاده از اشکال دو بعدی در یک پرسپکتیوه. در پرسپکتیو هر چه شئ به دوربین نزدیکتر باشه بزرگتر و هر چه دورتر باشه کوچکتر بنظر می آید. و بدین ترتیب وجود عمق رو تداعی می کنه.
در D3D برای ایجاد فضاهای سه بعدی توسط پرسپکتیو از تابع PerspectiveFovLH موجود در کلاس Matrix استفاده میکنیم. تین تابع اطلاعات مشخص شده در شکل زیر رو میگیره و ماتریس انتقال معروف به Projection رو برای استفاده در SetTransform بر میگردونه.



در شکل بالا زاویه مشخص شده با Fov همون زاویه دیده که با کم و زیاد کردنش می تونیم میدان دید رو تغییر بدیم.
فاصله پرده اول تا دوربین (NearPlane)با n و فاصله پرده دوم(FarPlane)با f نشون داده شده.
به ناحیه میان NearPlane و FarPlane ناحیه دید اتمی یا Frustum گفته میشه.واشیا برای دیده شدن باید در این ناحیه قرار بگیرن.
توضیح در مورد NearPlane و FarPlane :
فرض کنید در یک اتاق کاملا تاریک ایستادید.و از پنجره به بیرون نگاه می کنید.در بیرون هم چندین متر جلوتر یک برج مانع دید شما میشود.اگر فاصله شما تا پنجره n و فاصله شما تا برج رو f فرض کنیم،قادر به دیدن فضایی سه بعدی بین پنجره تا برج هستید.
و اگر شما رو دوربین فرض کنیم شکل بالا کاملا تشریح میشه.
البته تابع PerspectiveFovLH علاوه بر سه آرگومان فوق آرگومان دیگه ای به نام AspectRatio میگیره که از تقسیم طول صفحه نمایش به عرض اون بدست میاد. به عنوان مثال اگر در PP ابعاد صفحه نمایش ۶۰۰*۸۰۰ انتخاب شده Aspectratio برابر است با:

```
AspectRatio=(float)(800)/(float)(600);
const  float PI=3.14;
Matrix proj=new Matrix();
proj=Matrix.PerspectiveFovLH(PI*0.5,AspectRatio,1.0f,1000.0f);
device.SetTransform(TransformType.Projection,proj);
```

در تابع PerspectiveFovLH آرگومان اول زاویه دید بر حسب رادیانه که در اینجا 90 درجه انتخاب شده.آرگومان دوم AspectRatio است و آرگومانهای سوم و چهارم به ترتیب فاصله NearPlane و FarPlane رو از دوربین مشخص می کنند.
۸-ViewPortTransform:
این انتقال در موارد خاصی ضرورت پیدا میکنه که با رسیدن به اونها روش بحث می کنیم.
۹-Rasterization:
آخرین مرحله در رسم یک شئ کاملا به عهده D3D است. در این مرحله کلیه محاسبات لازم برای رسم یک شکل، از قبیل تنظیم لیست مثلثها،رنگ رئوس،نورافشانی و ... انجام میشه و شکل نهایی روی صفحه نمایش نشون داده میشه.

خسته نباشید.بالاخره این مقاله هم تموم شد.البته در این مقاله سعی داشتم تا تامی مسائل مربوط به رسم تصویر رو توضیح بدم تا از مقاله بعدی رسم اشکال رو شروع کنیم.
باز هم تاکید می کنم هر گونه سوال و یا مفهوم گنگ رو در تاپیک DirectX مطرح کنید.
