برمجة الالعاب بإستخدام الـ *DirectX* و *C++*

الجزء الأول

2007

Mohammed alaa

هذه النسخه غير منقحة

و قد توجد بها أخطاء

لماذا تلك الكورسات أفضل من غيرها ؟

برمجة الألعاب تجمع بين ثناياها مجموعة من العلوم الأخرى و التى لابد و أن تستوعبها جيداً قبل البدء فى كتابة الكود. و هذا لا يحدث مع الاغلبية العظمى من المبرمجين الذين يطمحون فى دخول ذلك المجال. فتجد أنهم يبدؤا التعلم بإسلوب الممارسة، فيقوم هذا الشخص بالبحث عن أكواد و يبدأ فى فهمها ثم ينتقل لكود أخر و هكذا. و حيث أن هذا الإسلوب ينفع و لكن و بالرغم أنه يأخذ قدر كبير من الوقت و المجهود لفهم أكواد قد تكون غير مرتبة أو بها اخطاء إلا أيضاً انه قد تؤدى لهذا الشخص لكراهية ذلك المجال لعدم فهمه لمجموعة من الأكواد!

الكورسات التى أقوم بتقديمها مرتبة بحيث أن اخر مرحلة نقوم بها هيا كتابة الأكواد، و لعدم الفهم الخاطئ لابد منك أن تعرف ما هى تلك المراحل (يتم تطبيق تلك المراحل على كل كورس على حدى):

1. نبدأ فى معرفة ما العلم المتعلق بالكورس و شرحه شرحاً وافياً من جميع الجهات كنظريات و مفاهيم و علاقتها بالألعاب.

2. الخطوة الثانية هى البدء فى تطبيق هذا العلم بإستخدام معادلات رياضياً و شرح كيفية أستخدام تلك المعادلات (و كيف تستطيع تطويرها لتناسب إستخداماتك).

3. البدء فى تطبيق ما عرفنا من الخطوتين الأولى و الثانية داخل لغة برمجه معينة (داخل تلك الكورسات سنستخدم لغة الـ ++C).

4. مشروع عملى تطبيقى على الخطوات الثلاثة السابقة.

كما تلاحظ فإنه عند وصولك للمرحله الثالثة سيكون لديك القدرة فى تطبيق ما عرفته بإستخدام اللغة التى تفضلها، لإنك حينها سكون لديك ما يؤهلك لذلك و سيكون لديك ما يجعلك تستطيع فيما بعد أن تطور نفسك بنفسك. لإنه حينها ستكون لديك مفاهيم و معارف أساسية.

لإى أستفسار أو معلومات عن الكورسات أرجو مراسلتى على :

Mohammed-alaa@hotmail.com
Mohammed1985@gmail.com

الدرس الأول : مبادئ الرسومات ثلاثية الأبعاد الجزء الأول

3D Graphics Fundamentals Part1

إسم الفصل : الفصل الأول (الصفحات من 45 - 1)

أهداف هذا الدرس :

نبدأ بالقارئ فى هذا الدرس بشرح الدوال الرياضية الأساسية لتصنيع الألعاب الثلاثية الأبعاد ، فى البداية سنتكلم عن الكائنات ثلاثية الأبعاد 3D Objects المستخدمة فى صناعة الألعاب و الممثلة فى الألعاب بشكل هندسى متعدد الأضلاع Polygonal Geometric وكيف ترسم بشكلها النهائى على شاشة الكمبيوتر ، من الهام جدا ان يستوعب الدارس المعادلات الرياضية الخاصة بعمليات التحويل للأشكال الثلاثية الأبعاد إلى أشكال ثنائية الأبعاد لكى يتم رسمها على الشاشة ، و من هذا المنطلق ننظر لتلك الأشكال المتعددة الأضلاع بشكل أعمق لنعرف كيف تتم عملية تحويل تلك الأشكال لترى على الشاشة بنفس الأسلوب التى ترى به وقت تصميمها أى ترى على الشاشة بشكل 2D دون ان تفقد شكلها الـ 3D ، و هذه الجزئية من الدرس تتضمن شرح إعادة تحجيم الكائنات Scaling و تدويرها Rotation و مفاهيم أساسية عن التنقل بين نظم الإحداثيات المختلفة مع إعادة رسم تلك الكائنات مرة اخرى على الشاشة بنفس الأحداثيات فى النظم الإحداثية المختلفة.

العناصر الرئيسية فى هذا الدرس :

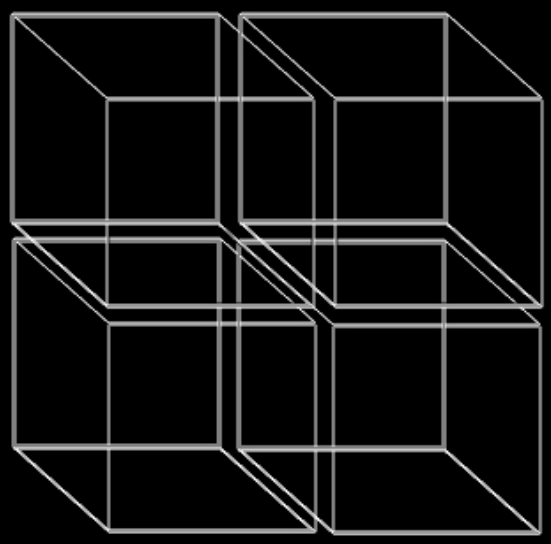
- الأشكال الهندسية متعددة الأضلاع
  - نظم الأحداثيات ثنائية و ثلاثية الأبعاد 2D/3D Coordinate Systems
  - المجسمات المعقدة Meshes
    - رؤس المجسمات الثلاثية الأبعاد Vertices
    - ترتيب الرؤوس Winding Order
- نظم التحويلات بشكل متقدم The Transformation Pipeline
  - نقل الإحداثيات Translation
  - تدوير المجسمات Rotation
  - منظور التحويلات Viewing Transformations
  - الرؤية من منظور طبيعى Perspective Projection
  - الرسم فى نظام إحداثيات الشاشة Screen Space Mapping

مشاريع مطلوبة لهذا الدرس : لا يوجد
أسئلة / أختبارات : لا يوجد
عدد ساعات الدراسة : من 5 إلى 7 ساعات

الفصل الأول

مبادئ الرسومات ثلاثية الأبعاد

الجزء الأول



© 2007, *Mohammed alaa*

الفهرس

العناصر الرئيسية
الأشكال الهندسية متعددة الأضلاع Geometric Modeling
المجسمات ثنائية الأبعاد Geometry in Two Dimensions
المجسمات ثلاثية الأبعاد Geometry in Three Dimensions
إنشاء اول مجسم Creating Our First Mesh
رؤوس المجسمات Vertices
ترتيب الرؤوس على الشاشة Winding Order
التحويلات Transformations
الرؤية من منظور طبيعى Perspective Projection
الرسم داخل إحداثيات الشاشة Screen Space Mapping
رسم المجسمات الأولية بإستخدام الكود الوهمى Draw Primitive Pseudo-code
المتجهات Vectors
طول المتجهات Vector Magnitude
جمع و طرح المتجهات Vector Addition and Subtraction
ضرب المتجهات العددية Vector Scalar Multiplication
وحدة المتجهات Unit Vectors
مضروب المصفوفات التربيعية The Cross Product
الممسات Normals
المضروب العددى The Dot Product
الإنزلاق Planes
المصفوفات Matrices
ضرب المصفوفات فى المصفوفات Matrix/Matrix Multiplication
ضرب المصفوفات فى المتجهات Matrix/Vector Multiplication
تدوير المصفوفات الثلاثية الأبعاد 3D Matrix Rotation
تعريف المصفوفات Identity Matrices
إعادة تحجيم و الحصول على قيمة المصفوفات Scaling and Shearing Matrices
دمج المصفوفات Matrix Concatenation
نظم الإحداثيات المتشابهة Homogeneous Coordinates
مجموع الأعداد الحقيقية و الكميات المتجهة Quaternion’s
الكائن D3DX
الكائن D3DXMATRIX
الكائن D3DXVECTOR3
الكائن D3DXPLANE
الكائن D3DXQUATRNION
دوال الكائن D3DX
التحويلات بشكل متقدم The Transformation Pipeline
المصفوفة الكلية The World Matrix
المصفوفة المحدودة The View Matrix
مصفوفة الرؤية من منظور طبيعى The Perspective Projection Matrix
الرؤية الحرة Arbitrary FOV
نسبة عرض الصورة التليفزيونية إلى طولها Aspect Ratio
خاتمة Conclusion

الدرس الأول

رقم الصفحة	العناصر الرئيسية
3	الأشكال الهندسية متعددة الأضلاع Geometric Modeling
5	المجسمات ثنائية الأبعاد Geometry in Two Dimensions
8	المجسمات ثلاثية الأبعاد Geometry in Three Dimensions
11	إنشاء اول مجسم Creating Our First Mesh
14	رؤوس المجسمات Vertices
15	ترتيب الرؤوس على الشاشة Winding Order
17	التحويلات Transformations
28	الرؤية من منظور طبيعى Perspective Projection
34	الرسم داخل إحداثيات الشاشة Screen Space Mapping
35	رسم المجسمات الأولية بإستخدام الكود المنطقى Draw Primitive Pseudo-code

الألعاب التى تستخدم الرسومات الـ 3D غالبا ما تستخدم مجموعة كبيرة من الملفات التى تستخدم فى كتابة الكود و التى تقوم بمجموعة من المهام مثل :

١. التعامل مع مدخلات المستخدم .
٢. تنظيم موارد النظام .
٣. تحميل الرسومات الـ 3D و إظهارها على الشاشة .
٤. ترجمة و تنفيذ الـ Scripts .
٥. تشغيل المؤثرات الصوتية .
٦. الذكاء الإصطناعى .

ملفات الأكواد هذه بالإضافة إلى العديد و العديد الأخرين يطلق عليهم مصطلح **محرك اللعبة** Game Engine ، أحد مهام محرك الألعاب و التى سوف يتم التركيز غليها بشكل قوى جداً فى هذا الكورس يطلق عليه **محرك الإكساء** Rendering Engine و يطلق عليه أسم أخر و هو Renderer ، و ظيفة محرك الإكساء تتلخص فى أخذ المعادلات الرياضية الخاصة بالعالم الخيالى الثلاثى الأبعاد و أظهاره فى شكل صور ثنائية الأبعاد ليتم إظهارها على الشاشة .

بعض المبرمجين المبتدئين فى عالم برمجة الجرفيكس يبدؤا مباشرة بدراسة الـ DirectX API الخاصة بالـ 3D بدون أى معرفة بسيطة بما تفعله هذه الـ API فى الخفاء ، و لذلك غالباً ما يسبب ذلك أخطاء جسيمة و غير متوقعة و طبعاً تستهلك من المبرمج وقت طويل فى تتبع الأخطاء و أكتشافها و تصليحها .

برمجة الرسومات الثلاثية الأبعاد تتضمن قدر كبير من الرياضيات ، بدون معرفة تلك الرياضيات بشكل قوى لن تستطيع و لن يكون لديك القدرة فى التعامل معها او اظهار كافة إمكانيات تلك الـ API .

و من هذا المنطلق ، فمن الخطأ كل الخطأ أن نبدأ فى دراسة برمجة الرسومات الثلاثية الأبعاد بدون تعلم مبادئ الرياضيات الخاصة بها ، لذا سنبدأ فى هذا الدرس فى دراسة تلك الرياضيات بالإضافة إلى بعض المفاهيم الخاصة بالرسومات الثلاثية الأبعاد .

فى هذا الفصل سوف نقوم بتصميم مشروع واحد فقط و الهدف منه هو تطبيق المعادلات الرياضية الخاصة بالـ 3D .

لإولئك الذين لديهم خلفية سابقة عن تلك الرياضيات ، نرجو أن لا تقوموا بتخطى هذا الدرس و إنما قرائته من باب التذكير و إنعاش الذاكرة .

خلال تصنيع و برمجة لعبة ثلاثية الأبعاد الرسامين و مصممى الجرافيك سيقومون بتصنيع مجسمات بإستخدام برامج قوية مثل GILES , 3D Studio Max , Maya .

تلك المجسمات ستستخدم و تحمل داخل العالم التخيلى للعبة ، فإذا أردت مثلا ان تصمم لعبة تحاكى الشارع الذى تعيش فيه ، سيقوم مصممى الجرافيك بتصنيع مجسمات عديدة للمنازل و ايضاً الشوارع و حتى الرصيف الذى نمشى عليه ، بعدها سيقوموا بتصميم مجموعة مختلفة من من المجسمات مثل مثل أعمدة الإنارة و الفوانيس التى تنير امام المنازل و السيارات و حتى الأشخاص ، هذه المجسمات سوف يتم تحميلها و إدخالها للعالم التخيلى للعبة حيث ان كل مجسم سوف يأخذ مكان و إحداثيات .

المجسمات غير المعقدة يمكن انشائها برمجياً بإستخدام معادلات و أساليب رياضية بسيطة ، و هذه هى الطريقة التى سنتبعها فى البداية مع الأمثلة التى سنأخذها ، هذه الطريقة سوف تساعدك على فهم كيف تكون المجسمات وقت الإنشاء و كيفية تمثيلها فى الذاكرة و كيف يتم إجراء العمليات عليها ، هذه الطريقة لإنشاء المجسمات تصلح فقط مع المجسمات البسيطة مثل المكعبات او الكرات أما مع المجسمات المعقدة فهذه الطريقة لن تكون إلا مضيعة لوقتك .

ملاحظة هامة : الكلمة 3D Models – المجسمات الثلاثية الأبعاد – يطلق عليها العديد من الأسماء بعضها مشهور مثل الكلمات : Objects و Models و Meshes ، و لتوحيد المصطلحات سوف نستخدم المصطلح Mesh لنعبر عن مجسم ثلاثى الأبعاد من مجرد مكعبي بسيط و حتى أعقد المجسمات .

الـ Mesh هو مجموعة من الأسطح Polygons تم توصيلها معاً لتكوين شكل خارجى مفهوم و يعرف بهيئته ، كل سطح داخل الـ Mesh يطلق عليه أسم Face – أى وجه – هذا الوجه يتكون من مجموعة من النقط Points موجوده فى نظام ثلاثى الأبعاد تم توصيلها على بإستخدام مجموعة من الخطوط ، و المفروض اننا نستطيع رسم السطح الموجود بين هذه الخطوط بأساليب عديدة سنذكرها كلما تقدمنا فى هذا الكورس ، على سبيل المثال البيانات المأخوذه من ملفات صور ثنائية الأبعاد يطلق عليها أسم **مكسيات** Texture Maps و تستخدم لإعطاء المجسم شكل و مظهر هذه الإكساءات انظر إلى الشكل التالى

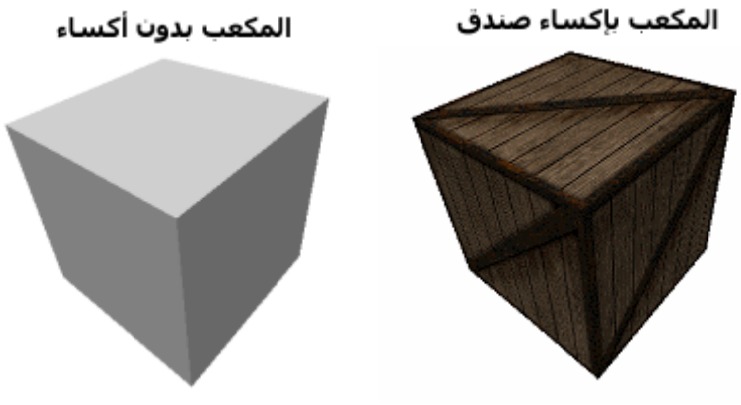


Fig 1.1

الـ Mesh الموجود فى الصورة السابقة يتكون من 6 أوجة أى يتكون من 6 Polygons و هما الوجه الامامى Front Face و الوجه الخلفى Back Face و الوجه الأعلى Top Face و الوجه السفلى Bottom Face و الوجه الأيمن Right Face و الوجه الأيسر Left Face بالنسبة للوجه الأمامى فهو طبعا يكون بناءاً على كيفية رؤيتنا للمكعب ، فى الواقع المفروض أن المكعب ثلاثى الأبعاد لذا يمكننا

أن نى 3 أوجه على الأكثر فى الوقت الواحد لإن الوجوه الثلاثة الأخرى تكون على الجانب الأخر من المكعب ، الصورة التالية ستوضح مفهوم الرؤئية للمكعب بشكل أوضح

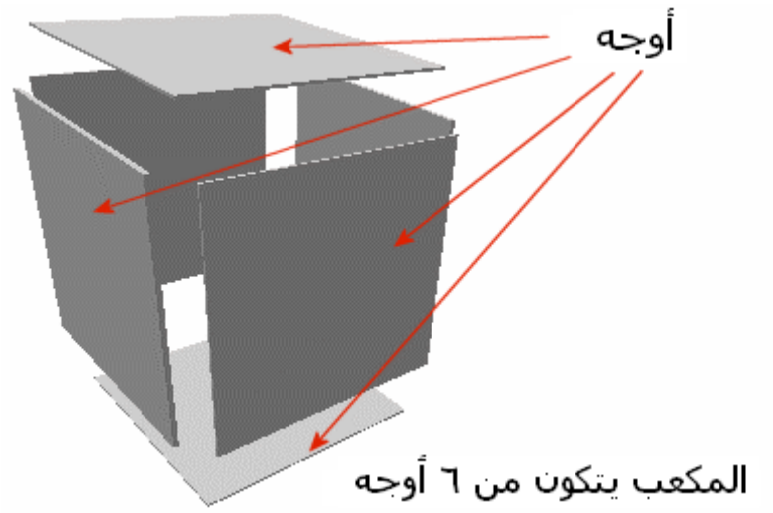


Fig 1.2

قوم بإنشاء وجه واحد Single Polygon ستقوم برسم مجموعة من النقط داخل نظام إحداثى ثلاثى الأبعاد ، الشكل الحقيقى لذلك الوجه سيظهر بشكل واضح عندما نقوم بتوصيل النقط معاً بمجموعة من الخطوط أنظر الشكل التالى

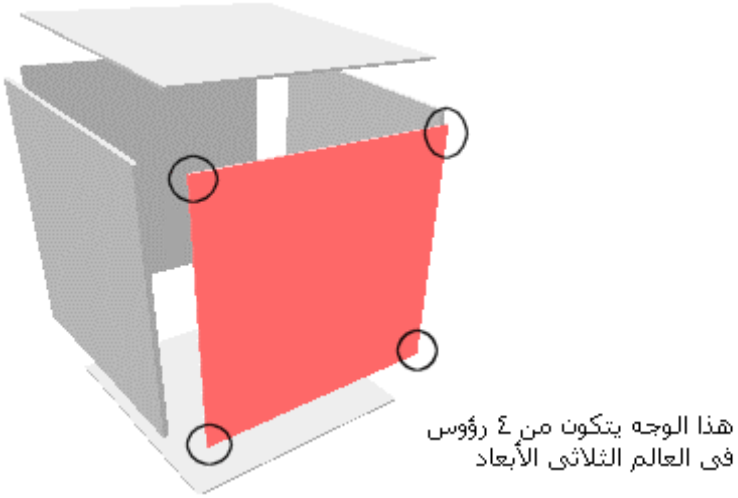


Fig 1.3

رسم النقط داخل نظام إحداثى معين و توصيلهم بعد ذلك بمجموعة من الخطوط يمكنا من إنشاء مجسمات ، هذا الجزء يعرف فى الرياضيات بإسم هندسة المجسمات Geometry ، الأن سوف نبدا فى دراسة الأشكال الهندسية فى عالم ثنائى الابعاد ثم سننتقل بعد ذلك لدراستها فى عالم ثلاثى الأبعاد .

نظم الإحداثيات Coordinate Systems هى إحدى خطوط الأعداد المستخدمة فى وصف العلاقات فى الفراغ .

كل خط من تلك الخطوط يسمى محور ، ففى حالتنا هذه – المقصود النظام الثنائى الأبعاد – يوجد محوران أحداهما أفقى و الأخر رأسى و يطلق عليهم X , Y ، المحور X هو المحور الأفقى أما المحور Y هو المحور الرأسى ، هذه المحاور تأخذ قيمة تبدأ فى التزايد بدءاً من نقطة الأصل Origin و نقطة الأصل هى نقطة تقاطع المحورين الأفقى و الرأسى و هى نقطة خاصة جداً حيث لا يوجد فى أى نظام أحداثيات أثنان منها و هى تأخذ الإحداثي (0 , 0).

فى النظام الثنائى الأبعاد كل النقط التى يتم رسمها على هذا السطح تأخذ إحداثى إستناداً إلى موقع تلك النقطة على السطح بالنسبة للمحورين و يكتب هذا الإحداثى فى الصورة القادمة (X , Y) .

الشكل التالى يعطينا مثال لنظام إحداثى ثنائى الأبعاد – و هذا النظام سنقوم بمعرفته معرفة تفصيلية لاحقاً فى هذا الدرس – هذا النظام يسمى نظام إحداثيات الشاشة Screen Coordinate System و هو يستخدم لوصف أماكن الـ Pixels فى المناطق التى نراها على الشاشة ، فى الشكل التالى المحور X يبدأ من اليسار لليمين و المحور Y يبدأ من أعلى لأسفل و نقطة الأصل بالنسبة للشاشة تكون فى الركن الأعلى على اليسار .

Fig 1.4

الشكل التالى يرينا إمكانية رسم أرع نقاط على الشاشة و كيفية رسم خطوط بينهم بشكل متتابع لتكوين شكل المربع ، هذا المربع يشبه كثيراً أحدى الأسطح المستخدمة فى صنع المكعب الذى صممناه من قبل مع الفارق اننا هنا نراه من منظور ثنائى الأبعاد و ليش ثلاثة أبعاد .

Fig 1.5

لابد و أن نرسم النقاط السابقة فى شكل متتابع معروف حتى نستطيع ان نرسم الخطوط بشكل واضح ، لقد رأينا أن هناك خط رسم بين نقطين 1 و 2 و خط أخر بين النقطتين 2 و 3 و هكذا بشكل متتابع حتى نقوم بتوصيل كل النقاط ببعضها و نكون وصلنا للنقطة رقم 1 .

نقطة هامة ألا و هى أن نظام إحداثيات الشاشة غير مفضل لتمثيل النظام الثنائى الأبعاد ، لإن المحور Y يزداد كلما تحرك لأسفل و هذا مخالف للواقع حيث التحرك لأسفل يقلل من قيمة الشئ و الأرتفاع لأعلى يزيد من قيمته و ليس العكس ، و نقطة أخرى و هى ان نظام الشاشة لا يسمح بالتعامل مع الأرقام السالبة او الأعداد التى تزيد عن أبعاد الشاشة .

فى الأنظمة المتكاملة المحورين X و Y يقوموا بالتعامل مع جميع الأرقام السالبة و الموجبة و فى الإتجاهات الأربعة المختلفة بدءاً من نقطة الأصل كما فى الشكل التالى

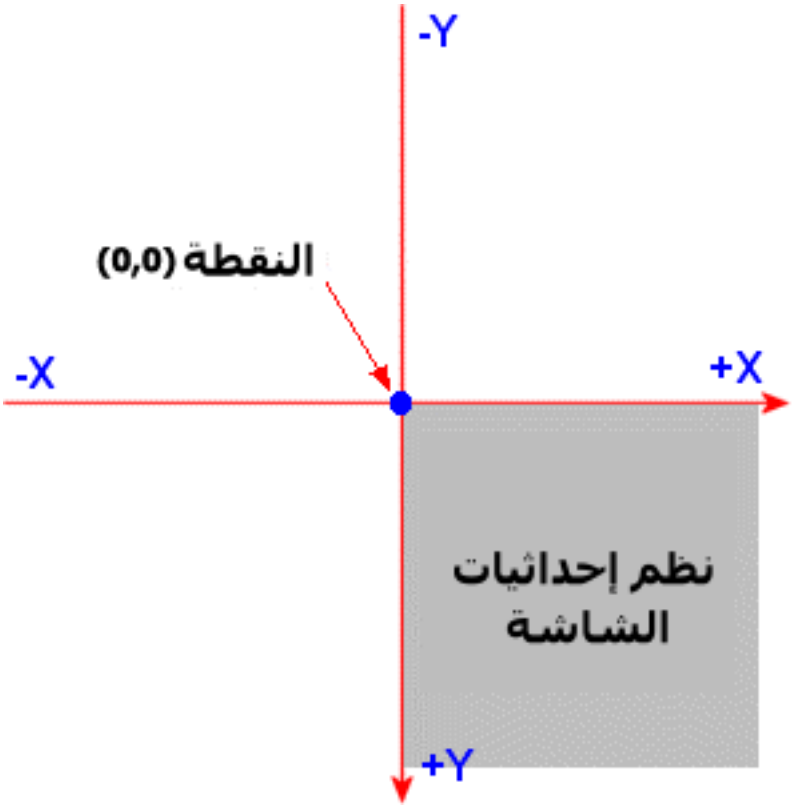


Fig 1.6

فى الشكل السابق جميع النقاط الموجودة فى داخل الربع السفلى لليمين للمحورين حيث يكونا X و Y موجباين هى الإحداثيات الصحيحة للشاشة ، الإحداثيات التى تقع فى اى من الـ ٣ مربعات الأخرى يتم تجاهلها تماماً .

نظام الإحداثيات الجديد سيقوم بتصحيح النظامين السابقين ، أولاً سيقوم بعكس أتجاه المحور Y و سيجعل الإتجاه الموجب لأعلى ، أيضاً سيدعم النظام الرقمى السالب و سيجعله لأسفل ، هذا النظام هو الأكثر أنتشاراً فى التعامل مع الأنظمة الثنائية الأبعاد و يطلق عليه أسم نظام الإحداثيات الديكارتى للأشكال ثنائية الأبعاد 2D Cartesian Coordinate System ، الشكل القادم يوضح مثال مرسوم فى النظام الديكارتى .

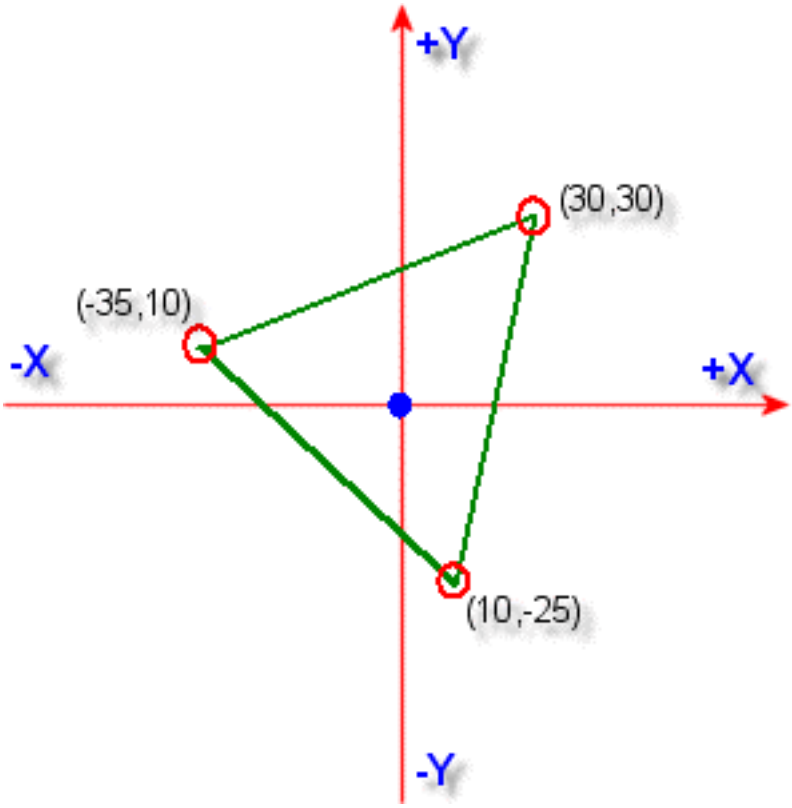


Fig 1.7

This Book maded by Mohammed alaa
Mohammed-alaa@hotmail.com

النظم الثلاثية الأبعاد تضيف البعد الذى يمثل العمق و هو المحور الثالث و يسمى Z أو Z-axis .

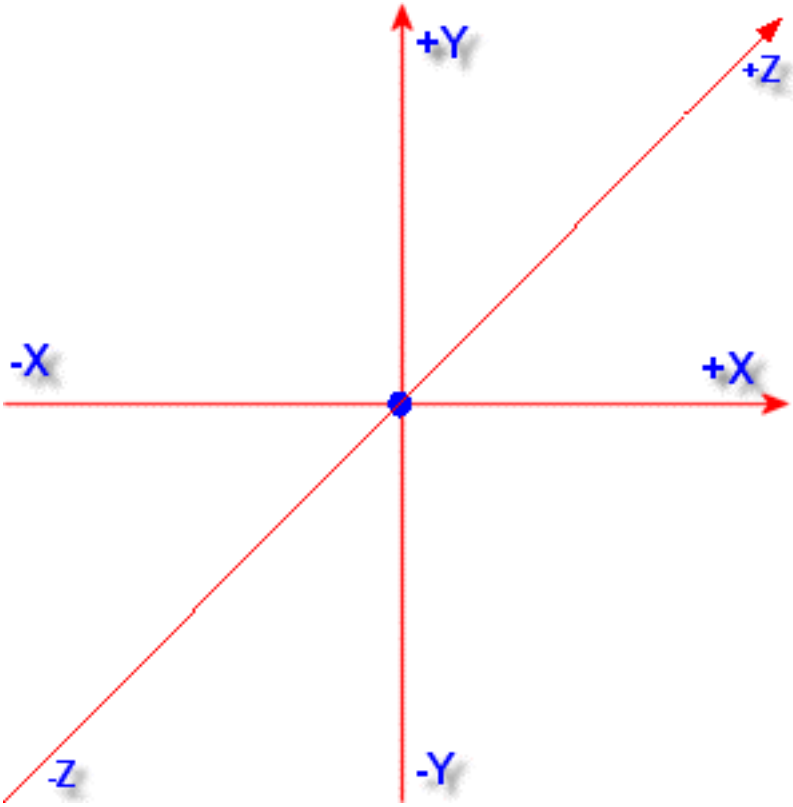


Fig 1.8

المحورين X و Y يبدؤا من نقطة الأصل لذا سنقوم بإضافة المحور Z إليهما و سيكون عمودى عليهما ، و من الرسم – التالى – نجد أن كلا المحاور الثلاثة عمودى على الأخر ، هذا فعلاً مطابق للواقع تماما حيث أن كل شئ بالإضافة إلى أن له طول و عرض له سمك أيضاً ، الصورة التالية ستوضح لك مفهوم العمق بشكل أفضل .

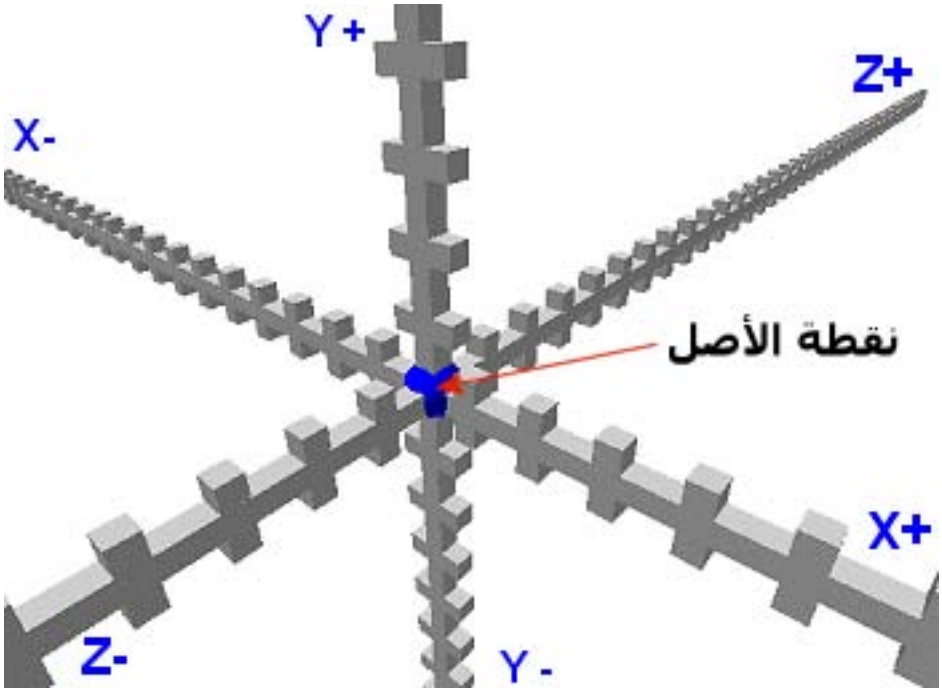


Fig 1.9

يوجد نوعين من النظام الإحداثى الديكارتى للأشكال ثلاثية الأبعاد و هما الأكثر أنتشاراً و أستخداماً ؛ إحداهما هو نظام اليد اليسرى – أو كما يعرف بإسم قاعدة اليد اليسرى – Left-Handed System و نظام اليد اليمنى – أو كما يعرف بإسم بقاعدة اليد اليمنى Right-Handed System ، الفرق بينهم هو الإتجاه الموجب للمحور Z .

فى نظام اليد اليسرى الحور Z يزيد كلما نظرت للأمام و يقل (أى يصبح سالب) من خلفك ، فى نظام اليد اليمنى يتم عكس الإتجاهين السابق ذكرهم ، بعض مكتبات الـ 3D مثل OpenGL تستخدم نظام اليد اليمنى ، فى حين أن مكتبات الـ DirectX تستخدم نظام اليد اليسرى و هى ما سنستخدمه إلى أنتهاء الكورس .

نظم الإحداثيات طبقاً لقاعدتى اليد اليمنى و اليسرى

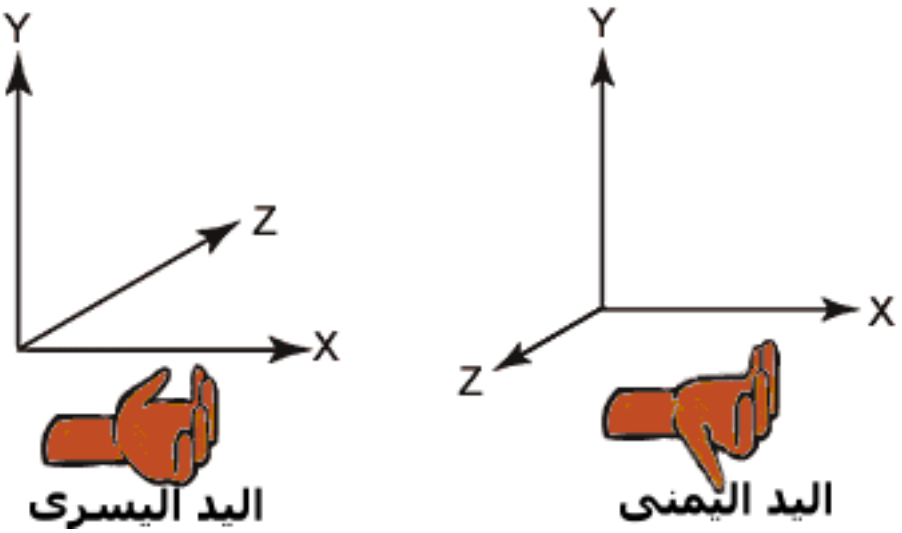


Fig 1.10

ملاحظة : لتتذكر إتجاه المحور Z فى النظام الذى تتبعه افعل الأتى :

١. أفرد ذراعيك الأثنين أمامك و هما يمثلوا الحور X .
٢. أجعل أصابعك الأربعة – الخنصر ، البصر ، الوسطى ، السبابة – مواجهة للسماء و هما يمثلوا المحور Y .
٣. ستجد أن أصبعك الأخير – الإبهام – قد توجه تلقائياً ليشير لإتجاء المحور Z و بناءاً على الذراع – من حيث كونها اليد اليسرى أم اليمنى – ستعرف أى الإحداثين تستخدم .

الأن ، لنقوم برسم نقطة فى النظام الإحداثى اليسارى لابد لنا من أن نحدد 3 أماكن للنقطة المستخدمة و هذه الاماكن تحدد أحداثيات المحاور X و Y و Z و نعبر عن هذه الأماكن رياضياً كالتالى (X , Y , Z) الشكل التالى يبين النقطة (1 , 2 , 2) فى هذا النظام .

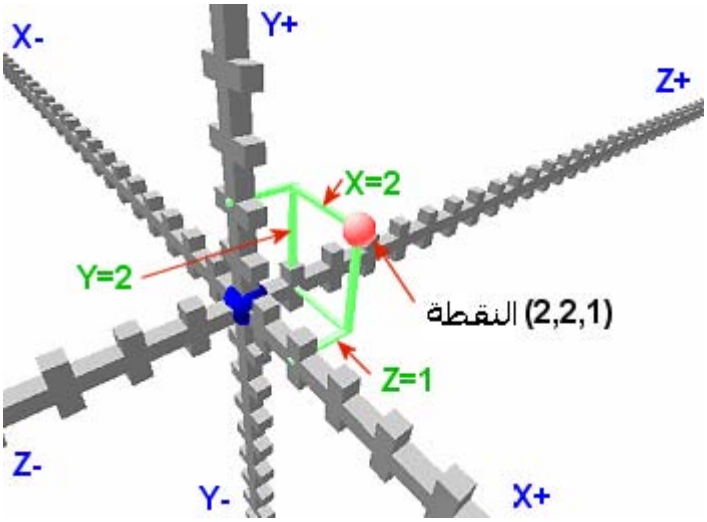


Fig 1.11

إن نظم الإحداثيات لها عدد لا نهائى من الأرقام ، و لكن بالنسة لك فإن نظام الإحداثيات مرتبط إرتباطاً كلياً بنوع المتغير الذى سيستخدم لحفظ أكبر رقم فى النظام الإحداثى – أى أكبر رقم على خط الأعداد للـ 3 محوار .

فمثلاً لو أخترت النوع float ليمثل قيم المحاور لذا من الممكن ان نعرف نقطة فى النظام الـ 3D كالتالى (8.0 , 65.2 , 1.0056) ،و لو أخترت النوع int بدلاً من السابق سنجد أن النقطة سيحدد إحداثيها أكثر من السابق كالتالى (22 , 64 , 1) .

فى أغلب محركات الإكساء يتم أستخدام النوع float لإن النوع int يسبب مشاكل فى العمليات الحسابية .

الكود المستخدم فى الأغلب لتمثيل نقطة فى الفضاء الثلاثى الأبعاد يأخذ الشكل التالى :

```
struct 3DPoint
{
        float x;
        float y;
        float z;
};
```

المجسم Mesh : هو مجموعة من الأسطح Polygons ، كل سطح يخزن فى الذاكرة على شكل نقطة ثلاثية الأبعاد ممثلة بالـ Data Type الذى قمنا بصنعه 3D Point .

فى حال اننا أردنا إنشاء مجسم لمكعب لابد و أن نحدد أماكن الـ 8 نقط المكونه له ، و كل سطح موجود فى المكعب مكون من 4 نقاط من هؤلاء الـ 8 .

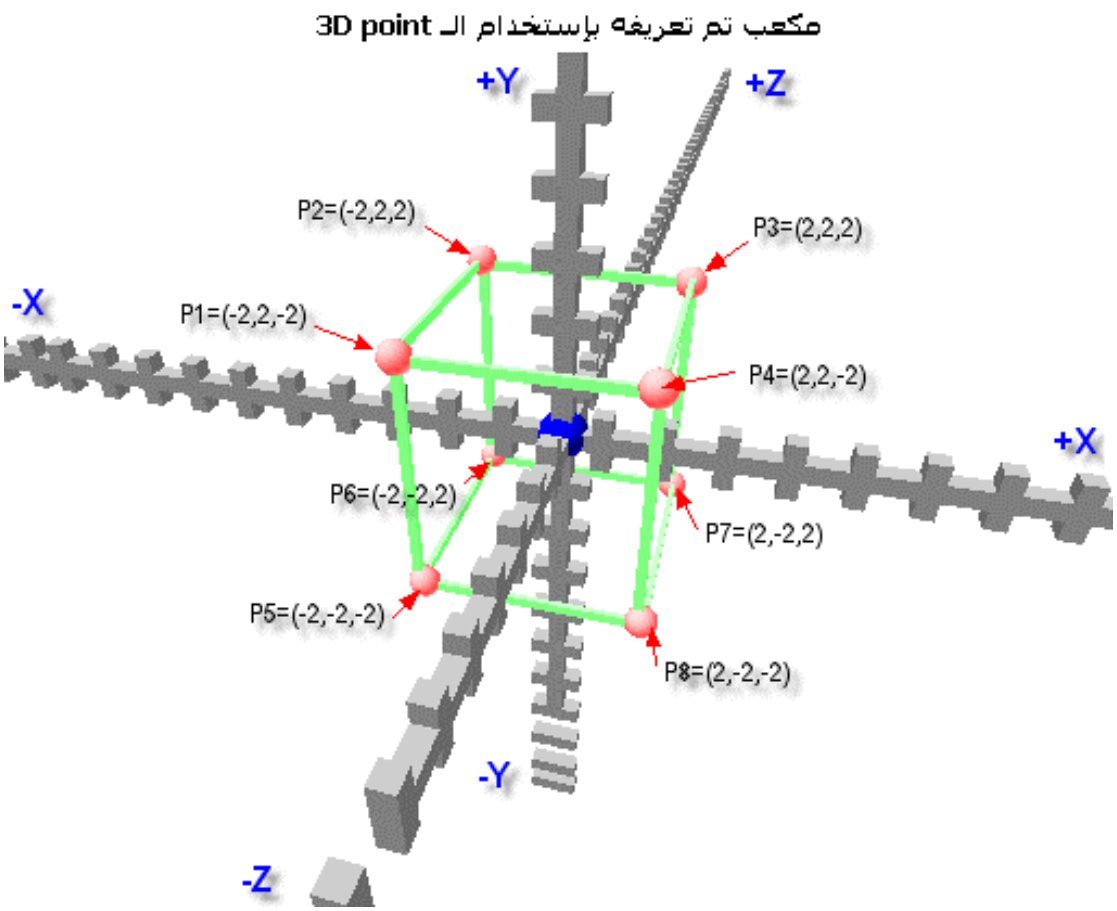


Fig 1.12

لاحظ أن مركز المكعب فى الرسم السابق موجود عند النقطة (0 , 0 , 0) و هى نفسها نقطة الأصل للنظام الأحداثى ، عندما يكون المجسم Mesh مركزه هو نقطة الأصل للنظام الإحداثى ، فى هذه الحاله يقال أن هذا الجسم موجود فى فضائه الخاص Model Space أو Object Local Space .

فى فضاء المجسم الخاص ، الأحداثيات تكون فى وسط المجسم التى هى بدورها نقطة الأصل الخاصة بالنظام الأحداثى ، لاحقاً فى هذا الدرس سوف نقوم بتحويل الإحداثيات الخاصة بالمجسم من فضائه الخاص – المساحة التى يشغلها هو فقط - إلى الفضاء الكلى – الذى يحتوى على المجسمات كلها – حيث تكون نقطة الأصل الموجودة فى نظام الإحداثيات ليست بالضرورة هى نقطة الأصل للمكعب .

ملاحظة هامة :

١. نقطة الأصل للمكعب تقع فى مركزه حيث تتقاطع أوتاره الأربعة .
٢. نقطة الأصل للنظام الإحداثى تقع حيث تتقاطع المحاور الثلاثة .

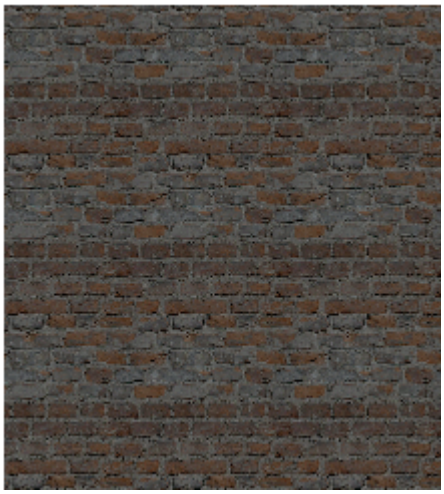
٣. فى الفضاء الكلى جميع المجسمات تشترك فى نفس نظام الإحداثيات و تشترك فى نقطة أصل واحدة و هى فى هذه الحالة مركز العالم التخيلى – اللعبة .

كثيراً جداً ما نريد أن نقوم بتدوير مجسم معين حول مركزه ، على سبيل المثال ، أنك تريد أن تجعل شخصية داخل اللعبة أن تدور حول مركزها حتى تغير من إتجاهها ، سنقوم بدراسة الرياضيات الخاصة بالتدوير لاحقاً فى هذا الفصل و لكن حتى الأن تذكر أنه إذا أردت أن تدور مجسم لابد منك و أن تدير جميع النقاط المكونه له ، على سبيل المثال لدينا مكعب و نريد تدويره بمقدار ٤٥ درجة لليمين كل ما علينا فعله هو تدوير النقاط المكونه له كلها بمقدار 45 درجة بإتجاه المحور Y .

عندما تدير نقطة فى نظام إحداثى معين ، مركز التدوير لتلك النقطة يكون دائماً و أبدا هو نقطة الأصل للنظام الإحداثى .

ملاحظة : وحدات القياس داخل اللعبة Note : Game World Units

تحديد وحدات القياس المستخدمة داخل اللعبة هو شئ يعود إلى المبرمج أولاً ثم على ما يتفق عليه مع الرسامين و مصممى الجرافيك الذين يعملون معه ، فعلى سبيل المثال ، قد تقرر ان وحدة واحدة تقابل واحد متر أى 1 Unit = 1 Meter و بعد ذلك تطلب من مصممى الجرافيك ان يصنعوا لك مجسمات تلائم هذا الحجم حتى تطابق المواصفات التى تريدها داخل اللعبة أو أن تحدد أن وحدة واحدة تقابل واحد كيلو متر أى 1 Unit = 1 Kilometer و بعد ذلك يصمموا لك المجسمات بناءاً على المعطيات الجديدة ، لذلك فمن المهم جداً أن تعلم أنه فى حال أختيارك وحدة قياس صغيرة للمجسمات فى لعبتك قد تواجه بعد ذلك مشكلة فى التعامل مع المتغيرات التى تحمل المساحات داخل اللعبة – وهى فى الأغلب تكون من نوع float ، المجسم الذى تتعامل معه من الممكن ان تكون ابعاده هى 4 * 4 * 4 مثل المكعب الذى تعاملنا معه سابقاً او حتى 100 * 100 * 100 و كون مثل السابق تماماً من منظور الشخص الذى يتعامل مع لعبتك ، لإنها تعتمد على برمجتك لها مثل السرعة المسموح للاعب ان يحرك شخصية معينه بها أو كيفية إكساء الوجوه faces ، فعلى سبيل المثال إذا نظرنا للصورتين التاليتين سنجد انهما متماثلتان معانهم استخدموا فى وحدات قياس مختلفة ، بالنسبة للسطح الموجود باليمين قد يكون ضخم جداً داخل اللعبة التى أستخدم فيها السطح الموجود إلى اليسار ، لذا فمهما كان عدد المجسمات داخل اللعبة الخاصة بك لابد و أن يكونوا قد صمموا بوحدات قياس ثابتة و بنسب ثابته حتى يكونوا متلائمين حين يظهروا بجانب بعضهم داخل اللعبة .



رؤوس المجسمات Vertices

الكلمة Vertex – و جمعها يكون Vertexes أو Vertices – هى عبارة عن مجموعة متكاملة من البيانات تصف نقطة واحدة داخل نظام ثلاثى الأبعاد .

من هذه اللحظة أى نقطة موجودة على سطح Polygon سوف تسمى بـ Vertex لإن هذا المصطلح أشمل و أعم ، لذا من الممكن أن نقول على المكعب أنه يحتوى على 24 رأس ، لإن المكعب يحتوى على ستة أسطح ، و كل سطح يحتوى على أربعة رؤوس لذا (4 * 6 = 24).

من منطلق أن المصطلح Vertex أعم و أشمل من المصطلح 3D Point – المذكور فى صفحة 11 – لذا فالكائن 3D Point الذى قمنا بإنشائه سابقاً لابد من تحديثه ، لذا فبرمجياً الـ Vertex سيمثل بـ Class لها الشكل التالى :

```
class CVertex
{
public:

        //Constructors
        CVertex (float fx, float fy, float fz);
        CVertex ();

        // Public Variable for this class
        float    x;   // vertex x coordinate
        float    y;   // vertex y coordinate
        float    z;   // vertex z coordinate
};
```

المجسمات المعقدة لن تصنع برمجياً و لكن سيتم صنعها بإستخدام برامج قوية مثل 3D Studio MAX , Maya , ... إلخ ، و هذا يتيح إنشاء مجسمات مكونة من ألالاف او ملايين الأسطح ، و هذا يؤثر سلباً على أداء اللعبة فكلما زاد عدد الأسطح كلما بطئت اللعبة لإنها تزيد من العمليات الحسابية اللازمة لرسمهم على الشاشة.

لذا فمهمتك كمبرمج ألعاب أن تستخدم مجموعة من الأساليب التى ستجعل عدد الأسطح التى سيتم رسمها فى وقت معين أقل ما يمكن. أحد الأساليب المتبعة هى عدم رسم أو أكساء الأسطح التى لن يستطيع المستخدم رؤيتها أبداً من موقعه الحالى من العالم التخيلى.

أسلوب أخر و هو تجاهل الأسطح التى تكون خلف المستخدم ، هذا الأسلوب يسمى بإخفاء السطح الخلفى Back Face Culling ، و هو يفترض أن المستخدم لن يسمح له بأن يرى ظهر سطح معين – مثل ما يوجد خلف حائط تنظر إليه – فى نفس الوقت الذى يرى فيه وجه ، أعتقد أنك لازلت تذكر المثال الخاص برؤية أوجه المكعب ، عندما قلت أنك لن ترى إلا 3 أوجه من الـ 6 أوجه الخاصة بالمكعب و هم فقط الين سيكونوا ظاهرين أمامك ، و لهذا السبب فمحرك الإكساء يقوم بعمليات حسابية سريعة و بسيطة قبل الإكساء ليرى هل الجزء الذى سيتم إكسائه يواجه المستخدم أم لا ن فإذا لم يكن يواجه فسوف يتم تجاهله.

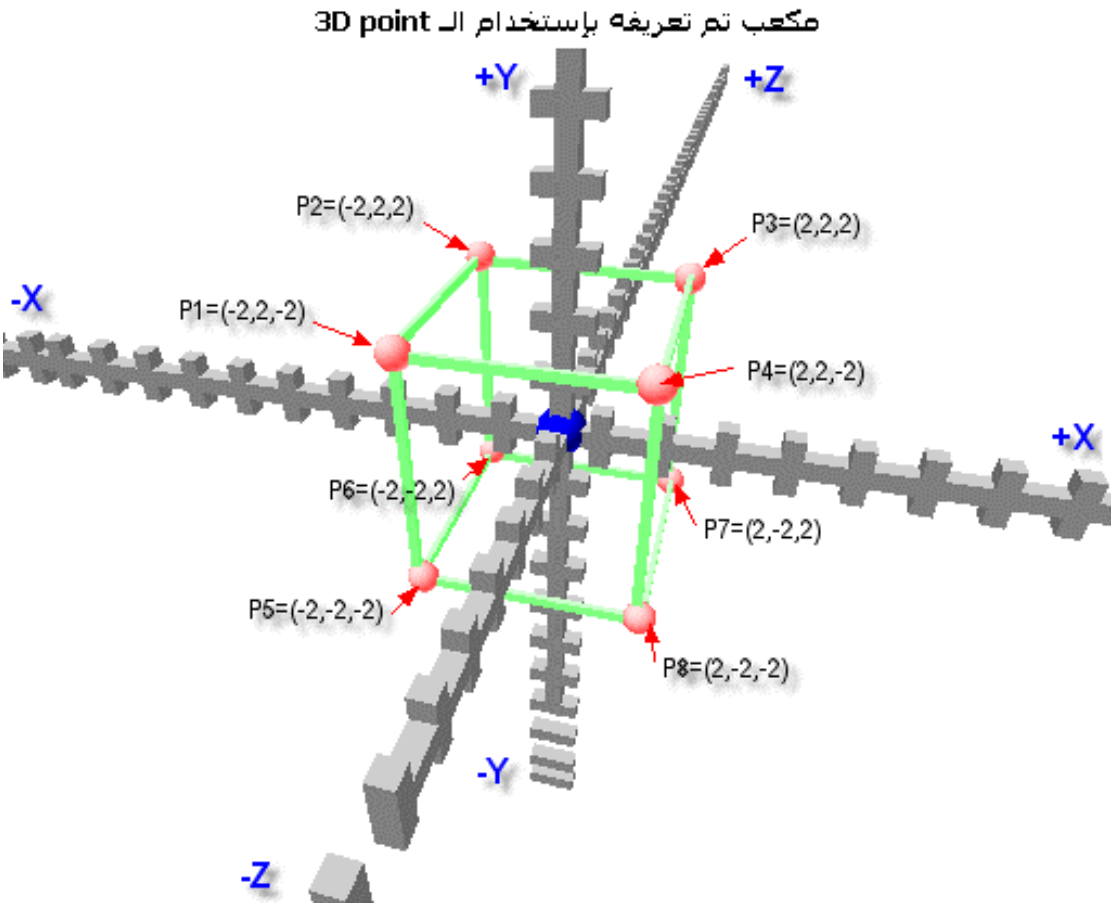


Fig 1.13

بإستخدام الصورة السابقة كمرجع ، لابد و انك ترى أن كل رأس موجودة على أى سطح من أسطح المكعب هى أحدى الرؤوس التى قمنا بتعريفها فى الكود الخاص بالـ Class التى أطلقنا عليها أسم CVertex.

الإحداثى P1 أستخدمت لإنشاء ثلاثة أوجه هما : الوجه الموجود باليسار و الوجه العلوى و الوجه المقابل لنا ... و هكذا لباقى الرؤوس ، لاحظ أن الرؤوس رسمت بإسلوب مرتب لتمكننا من رسم خط بين كل نقطتين متقابلتين على سطح واحد و هكذا حتى يكتمل السطح تماماً ، هذا الترتيب الذى أتبعناه يسمى بـ Winding Order.

التدوير فى أتجاه عقارب الساعة

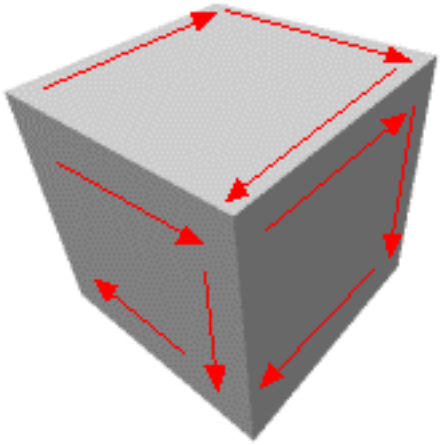


Fig 1.14

و لكن كيف يحدد شخص ما أى الأسطح تواجهه ؟

ببساطة و كما قلنا فى مثال المكعب ، الوجه يتكون من أربعة رؤوس و نحن لم نحدد إتجاه المكعب لذا فلإجابة تنطوى على كيفية إحتفاظنا بالرؤوس Vertices داخل الأسطح المكونة لهذا المكعب ، إذا نظرت للصورة رقم 1.13 ستجد أن الرؤوس تم رسمها و ترتيبها فى إتجاه عقارب الساعة.

على سبيل المثال ، الوجه الأمامى مكون من أربعة رؤوس هى P1 و P4 و P8 و P5 لذا عندما تنظر لذلك الوجه الأمام ستجد أن النقاط مرتبة فى إتجاه عقارب الساعة ، لذلك لا يهم أى الرؤوس الأربعة هى نقطة البداية لذلك السطح لإنك من الممكن أن تقول أن ترتيب الرؤوس كالتالى P8 و P5 و P1 و P4 ، كلا الترتيبين السابقين سليم و صحيح بنسبة 100% لإن كلاهما يتحرك فى إتجاه عقارب الساعه ، هذا الإسلوب فى ترتيب الرؤوس على الأسطح يعرف بـ Polygon Winding Order.

داخل مكتبات الـ DirectX الأسطح بشكل إفتراضى تعرف بترتيب فى إتجاه عقارب الساعة كما فى الشكل 1.14 و لكنك تستطيع تغييره وقتما تشاء.

لننظر الأن للسطح الخلفى ، و هو يحتوى على الرؤوس P6 و P7 و P3 و P2 هذا الترتيب كما هو واضح فى إتجاه عكس عقارب الساعة لذا فإننا لن نقوم برسمه ، طبعاً إذا قمنا بتدوير المكعب فسيكون السطح الخلفى مواجه لنا تماماً – أى سيكون هو السطح الأمامى – لذا ستجد أن النقاط موجودة فى إتجاه عقارب الساعة.

ملاحظة هامة :

الأسطح المواجهة للمستخدم تكون الرؤوس فيها دائما و أبدا فى إتجاه عقارب الساعة و الأسطح الغير مواجهه للمستخدم فى إتجاه عكس عقارب الساعة.

١. نقل الإحداثيات Translation

نقل الإحداثيات : هى أعطاء أحداثيات جديدة للرؤوس الموجوده على سطح معين داخل مجسم ما موجود داخل عالمنا التخيلى.

و سميت بنقل الإحداثيات Translation لإننا نقوم بنقل أسطح مجسم معين موجود داخل العام التخيلى من مكان إلى مكان و بمعدل ثابت.

فى الصورة التالية قمنا بتعريف سطح مكون من أربع رؤوس مركزهم حول نقطة الأصل لنظام الإحدثيات الخاص بالعالم التخيلى ، و بعد ذلك قمنا بنقل ذلك السطح من مكانه الحالى إلى مكان أخر داخل اللعبة و هذا المكان إحداثياته (0 , 5 , 0) ،لذا فبإضافة القيمة السابقة لكل الرؤوس سيتم نقل السطح من مكانه إلى مكان جديد داخل اللعبة و ستظل المسافة بين الرؤوس ثابته.

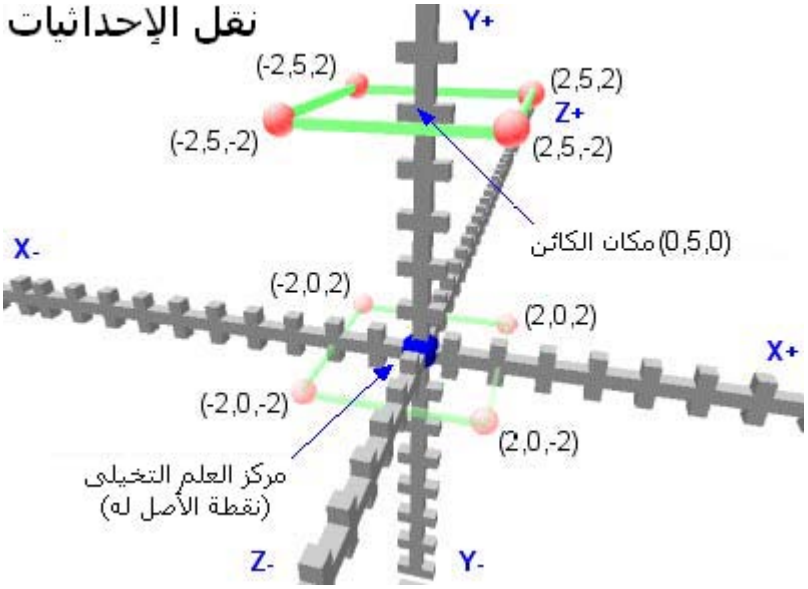


Fig 1.15

لنقوم بالتعبير عن العمليه السابقة بإستخدام الكود المنطقى

```
PositionInWorld.X=0;

PositionInWorld.Y=5;

PositionInWorld.Z=0;

For (Each Polygon In Mesh) {

        For (Each Vertex In Polygon) {

                Vertex.X += PositionInWorld.X;
```

```
            Vertex.Y += PositionInWorld.Y;

            Vertex.Z += PositionInWorld.Z;

        }

}
```

العملية السابقة يطلق عليها أيضاً إسم التحويل Transformation ، لإننا نقوم بتحويل إحداثيات المجسم من إحداثياته الخاصه Model Space للإحداثيات العامه – إحداثيات اللعبة – World Space.

الأن مركز السطح موجود بالإحداثى (0 , 5 , 0) فى اللعبة ، بنفس الإسلوب السابق يمكنك إطاء كل سطح و مجسم إحداثى خاص به داخل اللعبة.

لاحظ أننا لن نطبق الإسلوب السابق فى الكود الخاص بنا ، وبدلا منه سنقوم بتخزين النتيجة النهائيه داخل Class من نوع CVertex بشكل مؤقت حتى نقوم بإكسائها ، وبعد ذلك سنقوم بإستخدام كائن من نوع Mesh – سنقوم بصنعه لاحقاً – و هو يعبر عن المجسم الذى توجد به الرأس التى نتعامل معها ، هذا الكائن بياناته لا تتغير و يمكن ان تكون مشتركة مع بيانات كائنات أخرى موجودة داخل اللعبة ، هذه العمليه يطلق عليها إسم Instancing.

```
class CObject
{
public:

        CMesh     *m_pmesh;
        float      PositionX;
        float      PositionY;
        float      PositionZ;
};
```

فالنفترض اننا نريد ان يكون لدينا 3 مكعبات داخل لعبتنا ، لذا سنقوم بإنشاء 3 نسخ من الكائن CObject ، سنقوم بعدها بتحديد مكان كل نسخة عن طريق إعطاء قيم للمتغيرات PositionX ، PositionY و PositionZ. المؤشر m_pmesh الموجود بالـ 3 نسخ من الممكن أن يشير لنسخة واحدة من نوع CMesh ، لذا فلكل كائن موجود باللعبة الخاصة بنا سنتبع معه الخطوات التالية لإكسائه :

١. لكل سطح موجود على المجسم.
٢. قم بإضافة قيم الـ PositionX ، PositionY و PositionZ للمتغيرات X ، Y و Z الموجودين بالرؤوس.
٣. إحفظ النتيجة النهائية بشكل مؤقت فى مصفوفة من نوع CVertex.
٤. قم بإكساء الأسطح بإستخدام الرؤوس الموجودة فى الخطوة السابقة.

```
CMesh *MyMesh; // Pointer to the mesh containing our 4x4 polygon

CObject ObjectA, ObjectB, ObjectC;

ObjectA.m_pMesh = MyMesh;

ObjectB.m_pMesh = MyMesh;

ObjectC.m_pMesh = MyMesh;
```

ObjectA.PositionX = 0; ObjectA.PositionY = 5; ObjectA.PositionZ = 0;

ObjectB.PositionX = -6; ObjectB.PositionY = 0; ObjectB.PositionZ = 0;

ObjectC.PositionX = 4; ObjectC.PositionY = 0; ObjectC.PositionZ = -5;

فى منتصف الصورة القادمة – رقم 1.16 – نرى نسخة شفافة من للمجسم وقت وجوده فى فضائه الخاص ، بإضافة الإحداثيات الخاصة بالكائنات لرؤوس المجسم ، نقوم بهذا بنقل إحداثيات الكائن لمكان جديد داخل عالمنا التخيلى ، لاحظ ان المركز لكل كائن هو الذى يتحرك إلى مكان جديد ، و لأنه توجد مساحة ثابته بين كل رأس و مركز الكائن – أى نقطة الأصل الخاصه به – لذا فالرؤوس تتحرك حفاظاً على مكانها بالنسبة لمركز الكائن.

لاحظ الفرق بين كلمة مجسم و كلمة كائن ، فالمجسم هو الشكل الـ 3D الذى يراه اللاعب، أما الكائن فهى الـ Class التى تمثل هذا المجسم داخل الكود ، لذا فالكائن هو المسئول الأول و الأخير عن تحديث بيانات موقعه داخل اللعبة.

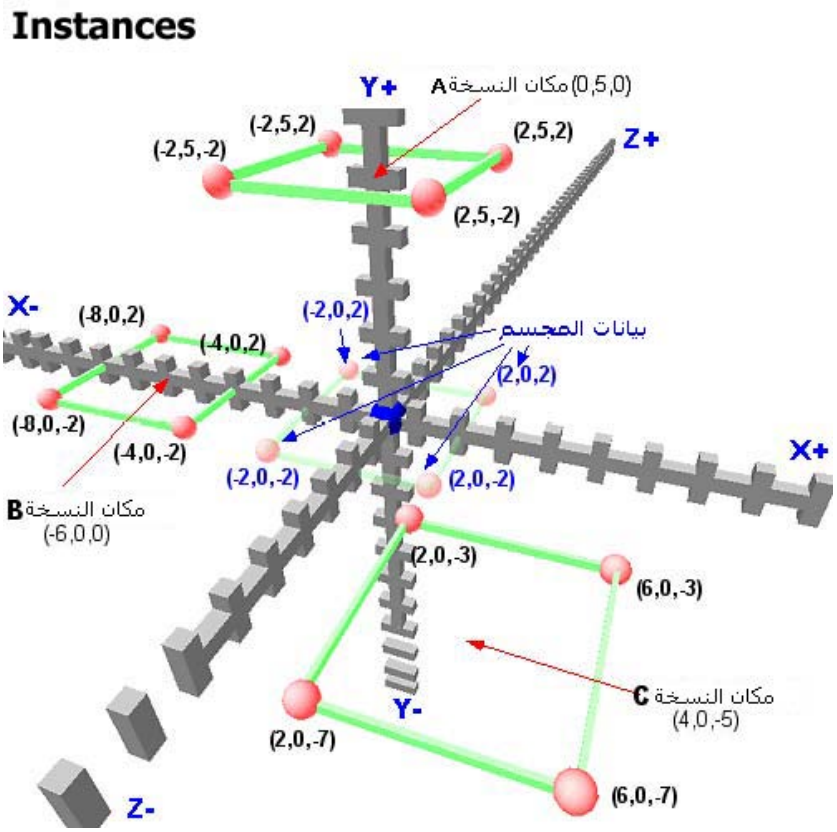


Fig 1.16

الدالة القادمة توضح كيف أن العمليات التى تتم على الكائن تؤثر على الـ Frames الخاصة باللعبة ،– الـ Frame يمثل الصورة الحالية التى تراها على الشاشة – لذا سنقوم بإعادة رسم جميع الكائنات من جديد حتى نرى التغيير الذى حدث ، الدالة DrawObjects تدور على جميع الكائنات و لكل سطح موجود داخل الكائن يتم إستدعاء الدالة DrawPrimitive لنقل السطح و إعادة رسمه و إكسائه.

```
void DrawObjects ()
{
	// transform vertices from model space to world space
	for (ULONG i = 0; i < NumberOfObjectsInWorld; i++)
	{
		CMesh *pMesh = WorldObjects[i]->m_pMesh;
		for ( ULONG f = 0; f < pMesh->m_nPolygonCount; f++ )
		{
			// Store poly for easy access
			CPolygon *pPoly = pMesh->m_pPolygon[f];

			// Transform and render polygon
			DrawPrimitive (WorldObjects[i], pPoly)
		}
	}
}

void DrawPrimtive (CObject * Object, CPolygon *pPoly)
{
	// Loop round each vertex transforming as we go
	for (USHORT v = 0; v < pPoly->m_nVertexCount; v++)
	{
		// Make a copy of the current vertex
		CVertex vtxCurrent = pPoly->m_pVertex[v];

		// Add world space position to transform to world space
		vtxCurrent.x += Object->PositionX;
		vtxCurrent.y += Object->PositionY;
		vtxCurrent.z += Object->PositionZ;

		// Do further pipeline transformations here which we have
		// not covered yet but will shortly.

		...
		...

		// By this point we will have 2D screen vertices so render
		// to screen which we have not yet Covered.
	}
}
```

التحويل من فضاء الكائن إلى فضاء اللعبة يحدث فى كل Frame لكل سطح نقوم بإكسائه ، بتحديد موقع الكائن داخل كل Frame بذلك نقوم بإنشاء حركة Animation ، فعلى سبيل المثال قد تجعل سفينة فضاء تتحرك داخل الفضاء عن طريق التلاعب بالزيادة و النقصان بالمتغيرات PositionX ، PositionY و PositionZ الموجودين داخل الكائن CObject المسئول عن السفينة.

٢.التدوير Rotation

لتقوم بتدوير مجموعة من رؤوس سطح موجود فى نظام إحداثى ثنائى الأبعاد ، سنقوم بإستخدام المعادلات التالية لكل نقطة على حدى :

$$NewX = OldX \times \cos(\theta) - OldY \times \sin(\theta)$$
$$NewY = OldX \times \sin(\theta) + OldY \times \cos(\theta)$$

فى المعادلات السابقة ، *OldX* ، *OldY* هما البعدين X ، Y للرأس التى سنديرها ، الدلتين Cos ، Sin هما دوال رياضيه و أسمائهما هى Cosine و Sine ، أما الرمز θ فيسمى بـ Theta و تمثل الزاوية التى سنقوم بإدارة الرأس إليها و تعطى قيمة لها بإستخدام نظام Radians – و هو نظام يسمى بالزاوية النصف قطرية و فيه تمثل الزاوية برقم عشرى – و ليس بنظام قياس الدرجات Degrees ، و السبب فى ذلك هو ان معظم مكتبات الـ 3D و منها الـ DirectX تستخدم النظام Radians لحساب الزوايا.

ملاحظة هامة :

الـ Radian يستخدم لتقدير قيمة الزاوية الحالية برقم عشرى ، و حيث أن الدائرة تنقسم لـ 360 درجه لذا فالـ Radian يقوم بقسم الدائرة إلى 2 * PI حيث PI تساوى 3.14159 وهى تقابل 180 درجة فى نظام Degree لقياس الزوايا ، لذلك يوجد تقريباً حوالى 6.28 Radian فى الدائرة الكاملة ، 90 درجة تقابل (PI/2) أى تساوى (1.1570795 Radian) وهكذا.

لإن أغلب المبرمجين يفضلوا التعامل مع نظام قياس الدرجات Degree ، لذلك يتم صناعة Macro ليقوم بتحويل القيمة من Degree إلى Radian كالتالى:

```
const float PI = 3.14159;

inline float DegToRad (float d) { return (d * ( PI / 180 ) ) }
```

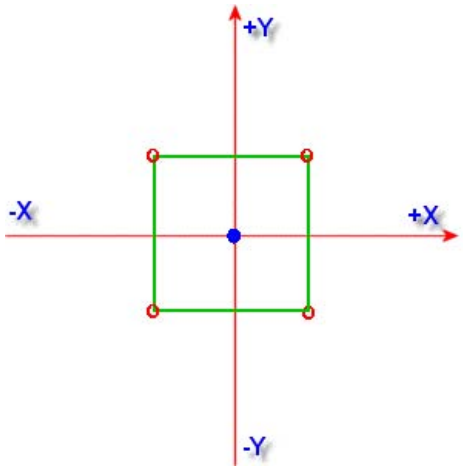


Fig 1.17

فى الصورة السابقة نريد ان ندير الأربع رؤوس الموجودين بها واحدة تلو الأخرى ، لذا فعن طريق معادلات التدوير السابقين و تطبيقهم على الرؤوس ستصبح الصورة كما فى الشكل القادم – الصورة 1.18 ، الكود التالى يوضح تلك العملية :

```
float angle = DegToRad( 45 );
for ( USHORT v = 0; v < pPolygon->m_nVertexCount; v++)
{
        CVertex OldVtx = pPolygon->Vertex[v];
        CVertex NewVtx;

        // Rotate the vertex 45 degrees
        NewVtx.x = OldVtx.x * cos(angle) - OldVtx.x * sin(angle);
        NewVtx.y = OldVtx.x * sin(angle) + OldVtx.y * cos(angle);

        // Vertex is now rotated and stored in NewVtx
        // Use to draw polygon in rotated position
}
...
```

قد تقول أن هذا التدوير يتم حول محور الـ Z ، تقنياً كلامك صحيح لإننا لا نرى محور الـ Z فى الصورة ، يمكنك أن تتخيل هذه الصورة أثناء وجودها فى عالم 3D ، فى هذه الحالة المحور Z لكل رأس يساوى صفر ، و المحور Z فى هذه الصورة سيكون يشير إلى الصفحة كما ذكرنا فى الجزء الخاص بالنظام الديكارتى للأشكال الـ 3D ، الصورة التالية هى نتيجة تدوير الأربع رؤوس بمقدار 45 درجة.

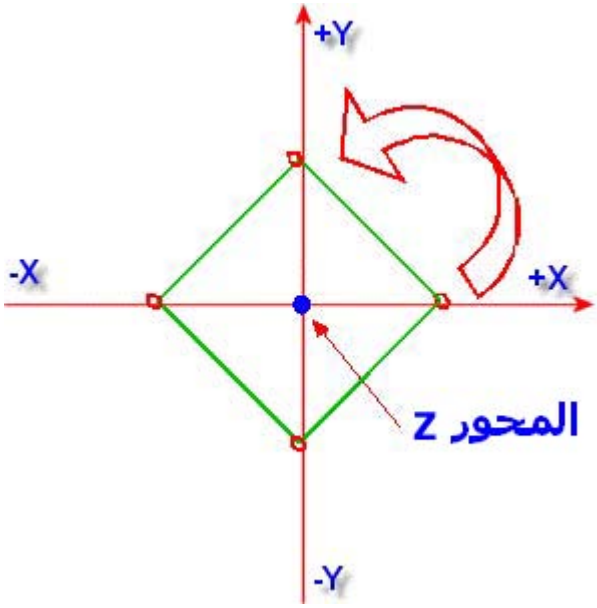


Fig 1.18

نقطة هامة لتتذكرها و هى فى أى نظام إحداثى التدوير يكون مرتبط بنقطة الأصل فمثلاً فى الصورة السابقة الرؤوس دورت حول نقطة الأصل – النقطة الزرقاء – و هذا هو مركز التدوير.

لاحظ أيضاً عندما تقوم بتدوير رأس حول محور معين ، فإن القيمة الخاصة بذلك المحور داخل الرأس – القيمة داخل المتغير الذى يمثل المحور – لا تتغير ، بمعنى أنه بتدوير الرأس حول المحور Y فقط

المحورين X و Z قيمتهم ستتأثر ، و أيضاً إذا قمنا بتدوير الرأس حول المحور X فقط المحورين Y و Z قيمتهم ستتأثر ، و أيضاً إذا قمنا بتدوير الرأس حول المحور Z فقط المحورين X و Y قيمتهم ستتأثر.

المعادلات التالية تستخدم لتدوير رأس فى فضاء 3D حول الـ 3 محاور :

١. التدوير حول المحور X

$$NewY = OldY \times \cos(\theta) - OldZ \times \sin(\theta)$$
$$NewZ = OldY \times \sin(\theta) + OldZ \times \cos(\theta)$$

٢. التدوير حول المحور Y

$$NewX = OldX \times \cos(\theta) - OldZ \times \sin(\theta)$$
$$NewZ = OldX \times \sin(\theta) + OldZ \times \cos(\theta)$$

٣. التدوير حول المحور Z

$$NewX = OldX \times \cos(\theta) - OldY \times \sin(\theta)$$
$$NewY = OldX \times \sin(\theta) + OldY \times \cos(\theta)$$

و حيث أن التدوير دائماً ما يكون مرتبطاً بنقطة الأصل لنظام الإحداثيات لابد و أن نكون حذرين فى الترتيب بين التدوير و نقل الإحداثيات ، فالنتخيل مثلاً أنه لدينا مجسم و نريد وضعة فى الإحداثى (0 , 5 , 0) و أيضاً نريد تدويره بمقدار 45 درجة حول المحور Z ، لذا قد نجرب الخطواط التاليه :

١. قم بتحريك رؤوس المجسم للنقطة (0 , 5 , 0) حتى ينتقل المجسم إلى تلك النقطة.

٢. قم بتدوير المجسم 45 درجه حول المحور Z.

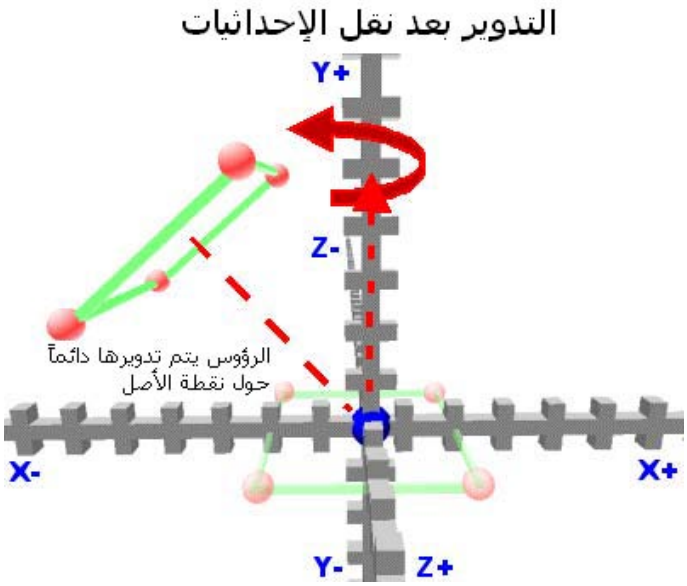


Fig 1.19

الصورة السابقة لا تظهر ما نتوقع لذا سأقول ماحدث ، أولاً تم تحريك المجسم للموقع (0 , 5 , 0) و بعد ذلك تم تدويره حول المحور Z و حول نقطة الأصل .

فى الأغلب و ليس دائماً ، ستقوم بتطبيق التدوير قبل نقل الإحداثيات لذا فالكائن سيتم تدويره داخل فضائه الخاص و حول نقطة الأصل الخاصة به ثم يتم تحريكه إلى مكانه الجديد داخل العالم التخيلى.

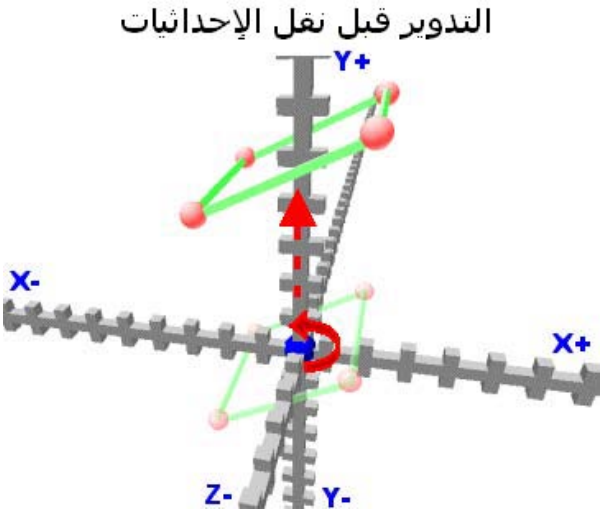


Fig 1.20

بتطبيق التدوير قبل نقل الإحداثيات سيكون لدينا القدره على نقل إحداثيات الكائن و هو موجود بزاوية 45 درجة حول مركزه – أى مركز الكائن -. بالطبع نقل الإحداثيات قبل التدوير قد يكون مفيداً فى بعض الحالات ، فمثلاً إذا كان لديك كوكب مركزه عند نقطة الأصل لنظام الإحداثيات فهنا يمكنك إستخدام هذا الإسلوب لتجعل كائن معين – جرم سماوى مثلاً – يدور حول هذا الكوكب بمسافة ثابته (مثل مدار).

سنقوم الأن بتطوير الكائن CObject بإضافة اعضاء جدد تسمح له بأن يدور حول محور معين.

```
class CObject
{
public:

        CMesh     *m_pmesh;

        float       PositionX , PositionY , PositionZ;

        float       RotationX , RotationY , RotationZ;
};
```

جميع تلك التحويلات ستحدث عندما نبدأ برسم مجسماتنا ، و هى ستحدث فى وقت تحضير الأسطح لرسمها على كل Frame ثم بعدها نبدأ برسم الأسطح.

٣.الكاميرات Cameras

قبل البدء فى رسم أى شئ لابد و أن تقوم بإنشاء كاميرا وهميه لتقوم من خلالها برؤية العالم التخيلى ، بعد ذلك جميع الرؤوس الموجودة بالعالم التخيلى يتم رسمها بناءاً على موقعها بالنسبة للكاميرا ، هذا يتطلب نظام إحداثى جديد يسمى نظام الكاميرات Camera Space أو منطقة الرؤية View Space ، و كما رأينا سابقاً ، لابد من أن نعيد تحويل رؤوس المجسمات لتتناسب مع النظام الإحداثى الجديد. سيتطلب منا مع نظام الكاميرا الجديد أن نحديد بعض الخصائص مثل موقع الحالى للكاميراCamera Current Position ، إتجاه الرؤية Viewing Direction و أيضاً مجال الرؤية الخاص بالكاميرا (FOV) Field Of View.

يفترض ما ان نستطيع توجيه الكاميرا داخل اللعبة حتى نستطيع الرؤية بشكل حر ، و هذا يتم ببعض الخطوات كالتالى :

١. عندما يقوم اللاعب بتحريك الكاميرا للأمام ، نقوم بتحريك العالم كله للخلف.
٢. عندما يقوم اللاعب بتحريك الكاميرا للخلف ، نقوم بتحريك العالم كله للأمام.
٣. عندما يقوم اللاعب بالإلتفاف لليسار حول المحور Y ، نقوم بإدارة العالم كله لليمين حول المحور Y.
٤. وهكذا.....

كما نرى ، أياً كان ما نريد أن نجعل الكاميرا تفعله ، لابد و أن نجعل كل كائن داخل عالمنا أن يفعل عكسه ، و هذا يعطينا الإحساس بالحركة داخل العالم التخيلى ، و لكن فى الحقيقة العالم هو الذى يتحرك حولنا.

سنقوم الأن بصناعة Class لتمثل الكاميرا و هى ستحتوى فقط على موقع الكاميرا داخل العالم و زوايا إدارة الكاميرا :

```
class CCamera
{
public:
        float       PositionX , PositionY . PositionZ;

        float       RotationX; // Pitch
        float       RotationY; // Yaw
        float       RotationZ; // Roll
};
```

حيث :

١. Pitch : تمثل التحرك لأعلى و أسفل
٢. Yaw : تمثل التحرك لليمين و اليسار
٣. Roll : تمثل التحرك للأمام و الخلف

قد نضيف فى الدالة السابقة دالة لتحويل حركة الماوس او الـ Joystick إلى درجات لتسجل فى المتغير Rotation ، فمثلاً التحرك للسار من خلال الـ Joystick قد نزيد درجة واحدة داخل المتغير RotationY لتجعل الكاميرا تدور لليسار ، إذا ضغطنا زر الأمام حينها سنقوم بزيادة قيمة المتغير RotationZ لنجعل الكاميرا تتحرك للأمام داخل العالم.

الكود القادم يقوم بعمل إكساء لكل الرؤوس بناءاً على موقع الكاميرا الحالى :

```
void DrawObjects()
{
        for (each object)
        {
                for (Each Polygon in Object)
                {
                        DrawPrimitive (Object, Polygon, Camera);
                }
        }
}

void DrawPrimitive (CObject * Object , CPolygon *Poly , CCamera * Cam)
{
for (each Vertex in Poly)
        {

        // convert polygon to world space (Already discussed)
        Perform any object Rotations on vertices of polygon
        Perform Translation on vertices of polygon to move polygon into world space

        // convert polygon to view space
        Perform inverse camera rotations on vertices of polygon
        Perform inverse camera translations on vertices of polygon

        // Convert polygon to Projection Space
        Not Yet Covered
        // Render 2D polygon
        Not Yet Covered
        }
}
```

لو كان المحور X للكاميرا يحتوى على قيمة 45 درجة لذا فالكود القادم سيقوم بتدوير كل الرؤوس الموجودة بكل كائن موجود فى العالم التخيلى بمقدار 45- درجة حول المحور X (أى 45 درجة فى الإتجاه المضاد للكاميرا).

```
if (m_pCamera.RotationX)
{
        VSVertex.y = Vertex.y * cos(-m_pCamera.RotationX) –
                     Vertex.z * sin(-m_pCamera.RotationX);
        VSVertex.z = Vertex.y * sin(-m_pCamera.RotationX) +
                     Vertex.z * cos(-m_pCamera.RotationX);
} // End if X Axis Rotation
```

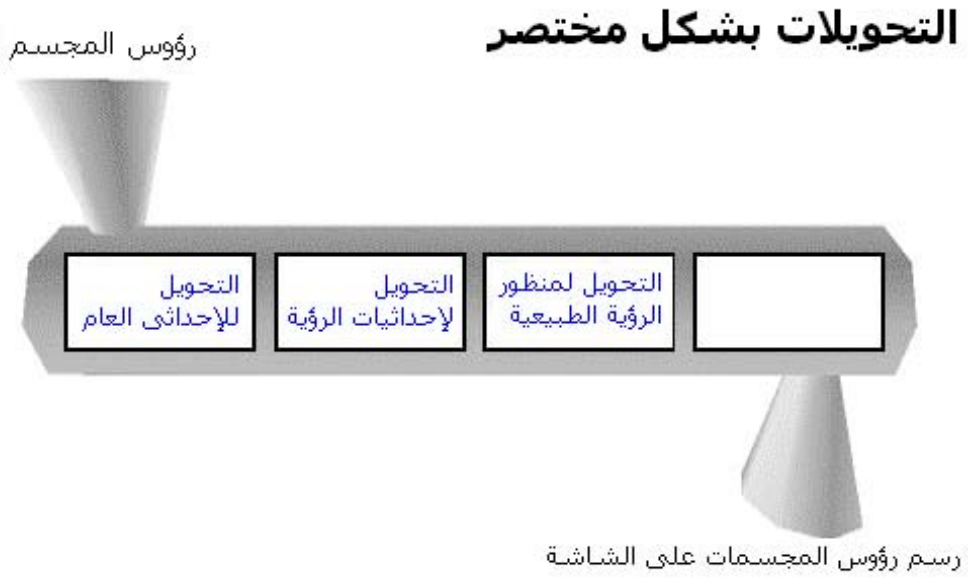
التدوير حول المحورين Y و Z سيكون مشابهاً للكود التالى :

```
if (m_pCamera.RotationY)

{

        VSVertex.x = Vertex.x * cos(-m_pCamera.RotationY) –

                     Vertex.z * sin(-m_pCamera.RotationY);
```

```
        VSVertex.z = Vertex.x * -sin(-m_pCamera.RotationY) +
                     Vertex.z * cos(-m_pCamera.RotationY);
} // End if Y Axis Rotation
if (m_pCamera.RotationZ)
{
        VSVertex.x = Vertex.x * cos(-m_pCamera.RotationZ) +
                     Vertex.y * sin(-m_pCamera.RotationZ);
        VSVertex.y = Vertex.x * sin(-m_pCamera.RotationZ) +
                     Vertex.y * cos(-m_pCamera.RotationZ);
} // End if Z Axis Rotation
```

فى الكود السابق ، المتغير Vertex يفترض به أن يكون فى مجال الإحداثى العام للعبه World Space و يتم تغييره ليصبح فى مجال الكاميرا أو مجال الرؤيه ، نفس معادلات المذكورة سابقاً تم إستخدامها هنا مع جعل الزوايه تأخذ قيمة سالبه – لجعلها فى الإتجاه المضاد ، أيضاً لو كان للكاميرا إحداثى فى الـ World Space غير (0 , 0 , 0) ففى تلك الحالة سيتم تطبيق المعادلات السابقة عليها كما سنرى لا حقاً فى هذا الدرس.

إنه لجدير بالذكر أن تعلم أنه تم تخصيص فصل كامل فى هذا الكتاب ليتناول شرح انظمة الكاميرات بشكل مفصل و أكثر دقة ، لذا لا تقلق حيال غموض او عدم فهمك تجاه المعادلات السابقة لإنه سيتم تناولها بشكل أكثر تفصيلاً فى ذاك الفصل.



منظور الرؤية من المواضيع الهامة فى كيفية تحديد المسافات و القياسات فى العالم حولنا ، داخل العلم الحقيقى ، كلما تحرك الشئ لمسافة بعيدة كلما قل حجمة و العكس بالعكس. و بالمثل سوف نفعل بمجسماتنا داخل عالمنا التخيلى ، فكلما بعدت الكاميرا عن المجسمات ، فلابد و أن تنكمش Shrink – أى يقل حجمها ، و عندما نقترب منها لابد و أن تكبر Grow .

و أيضاً عندما تتجه المجسمات للأمام حيث مركز رؤيتك فعندها تزيد المسافة بينك و بينها – أى تبعد عنك ، و تبعد المجسمات عن مركز الرؤية كلما قلت المسافة بينكم – أى تقترب منك ، أغلب المجسمات و التى تظهر داخل مجال الرؤية ستجد أنها تحركت للأمام او للخلف او لليمين أو لليسار بما يتناسب مع مكانها من مركز الرؤية عندما تقترب منك ، و عند حدوث ذلك يمكنك معرفة قيمتى الإتجاهين اليمينى و اليسارى عن طريق معرفة قيمة المحور X ، كذلك يمكنك معرفة قيمتى الإتجاهين الأعلى و الأسفل عن طريق معرفة قيمة المحور Y بما يتناسب مع محور الرؤية .

فى أحد الأجزاء السابقة ، عرفنا كيفية تحويل رؤوس المجسمات لمجال رؤية الكاميرا حيث أماكن الرؤوس تتناسب مع نظام إحداثيات الكاميرا ، كل ما نفعله هو طلب المسافة من موقع اللاعب الحالى لأى كائن موجود داخل العالم التخيلى و بعدها نستنتج قيمة المحور Z لمكان اللعب .

كلما زادت المسافة بين اللاعب و ذاك الكائن نقوم حينئذ بإعادة تحجيم الرؤوس الخاصة بذلك الكائن بكمية معينة - و هى المسافة المعطاه من اللاعب – حول المحورين X ، Y تأثير الرؤية المطلوب .

$$2DX = \frac{ViewSpaceX}{ViewSpceZ}$$

$$2DY = \frac{ViewSpaceY}{ViewSpceZ}$$

فى المعادلات السابقة ، نقوم بقسمة أماكن الرؤوس الخاصة بالمحورين X ، Y على مسافة الرؤية و هى قيمة المجور Z ، لذلك تكون النتيجة عبارة عن نقطة فى مجال رؤية ثنائى الأبعاد .

فالنتخيل مثلاً أننا لدينا إحداثى لرأس ما يقابل 5 وحدات ليمين الكاميرا ، 20 وحدة اعلى الكاميرا و 100 وحدة أمام الكاميرا ، لذا الأرقام السابقة يمكن ان تكتب بشكل رياضى كالتالى (5 , 20 , 100) و تلك الإحداثيات توجد داخل مجال الكاميرا .

سنقوم الأن بتطبيق معادلات الرؤية السابقة :

$$2DX = \frac{ViewSpaceX}{ViewSpceZ} = \frac{5}{100} = 0.005$$

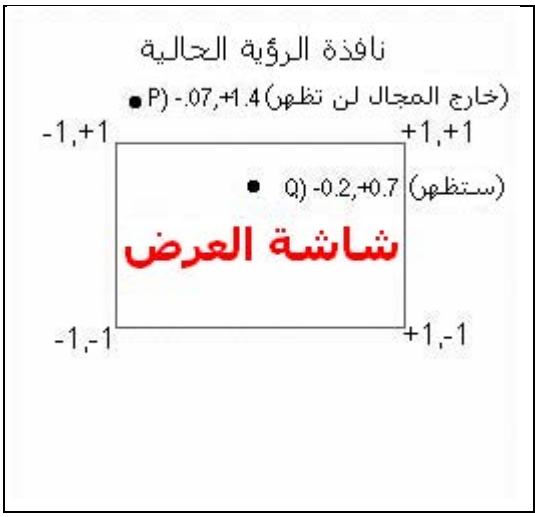
$$2DY = \frac{ViewSpaceY}{ViewSpceZ} = \frac{20}{100} = 0.2$$

النتيجة السابقة هى نقطة فى مجال رؤية ثنائى الأبعاد و إحداثيتها هى (0.005,0.2).

لاحظ أن الإحداثيات السابقة ليست إحداثيات الشاشة و بما أننا تطرقنا لهذه النقطة لذا فلابد من الأرقام السابقة و أن تكون أرقام صحيحة فقط .

نظام الإحداثيات الذى قمنا بإتباعه فى المعادلات السابقة يسمى بنظام الرؤية المحدودة Viewport Space Coordinate و فى بعض الأحيان الأخرى يطلق عليه أسم نظام الرؤية المختصر Clip-Space Coordinate و أسم نظام عرض الصورة المتحركة على الشاشة Projection Space Coordinate .

النظام الإحداثى الثلاثى الأبعاد الذى نتعامل معه تم تحويله إلى سطح ثنائى الأبعاد لا نهائى ، فى هذا السطح توجد منطقة يطلق عليها أسم نافذة الرؤية الحالية أو نافذة العرض Projection Window ، إذا كانت قيمة المحورين X ، Y اللذان قمنا بإستنتاج قيمهما من المعادلات السابقة تقع داخل نطاق هذه النافذه لذا سيكونوا ظاهرين أمام الكاميرا و لابد من رسمهم و إكسائهم ، عند هذه المرحله تقوم مكتبات الـ DirectX بإجراء أختبار لترى هل تلك الأسطح خارج مجال رؤية المستخدم – إخفاء السطح الخلفى – فإذا كانت خارج مجال الرؤية لذا يتم إظهار الأسطح المواجهة للمستخدم .



١) نظام عرض الصورة المتحركة تم تحجيمه بإستخدام نافذة الرؤية الحالية عن طريق المعادلات x/z و y/z .

٢) النقطة P ليست داخل نافذة الرؤية لذا لن يتم رسمها .

٣) النقطة Q داخل نافذة الرؤية لذا سيتم رسمها و إكسائها .

نافذة الرؤية فى أخر الصفحة السابقة هى شاشة مربعة ثنائية الأبعاد و أبعادها وحدتين طول و وحدتين عرض و نقطة الأصل فى مركزها لذا فنقطة الأصل ستقع داخل مركز شاشة العرض ، إحداثيات الشاشة السابقة تقع بين 1- و 1+ للمحورين X و Y .

ملاحظة هامة :

الإحداثيات السابقة – الموجودة داخل الصورة للنقطتين P و Q – تم إيجاد إحداثياتهما بعد قسمة قيمة المحورين X و Y على قيمة المحور Z

فى المعادلات المذكورة فى المثال السابق ، قيمة المحورين X و Y كانتا (0.2 , 0.005) و هما تقعا داخل المدى 1- و 1+ ، لذا فتلك النقطة تقع داخل مجال نافذة الرؤية لذلك ستكون تلك النقطة ظاهرة للكاميرا (داخل مجال الرؤية) .

الصورة القادمة ترينا الرؤية من أحد جوانب الكاميرا داخل نافذة الرؤية .

من منظور الصورة التالية من الهام أن تعرف أنها تفترض أن المحور X يشير لداخل الصفحة لذلك لا يمكن رؤيته و هذا المنطق يطبق أيضاً على أنظمة العرض للإحداثى X.

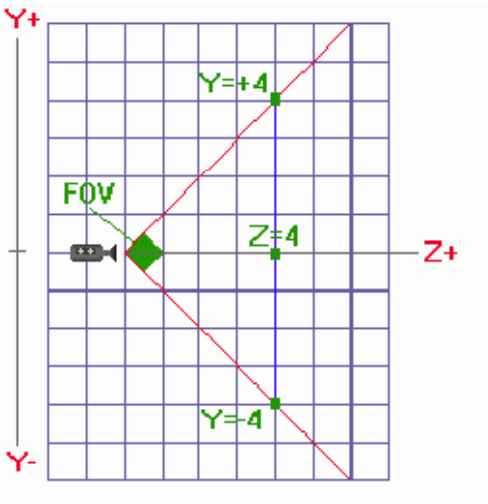


Fig 1.21

فى الصورة السابقة كلما زاد المحور Z كلما تم تحجيم المحور Y معه بنسبه ثابته ، فمثلاً لدينا نافذة رؤية أبعادها القصوى بالنسبة للمحور Y هى 1+ ، لذا ففى حال أن المحورين Y و Z متساويين أى Y=Z لذا فمعادلات الرؤية ستكون كالتالى :

$$Y = \frac{Y}{Z} = \frac{Y}{Y} = 1$$

فى حال أن Y=Z داخل مجال الرؤية لذا سنجد أن تلك النقطة سترى من أعلى مكان داخل نافذة العرض .

كلما زاد المحور Z سنجد أن أعلى قيمة يأخذها المحور Y ستقع دائماً داخل نافذة العرض و هى Y=Z أو Y=-Z ، نفس النتيجة صحيحة بالنسبة للعرض من خلال المحور X .

لذا فلأى نقطة فى حال أن Y>Z أو Y<-Z أو X>Z أو X<-Z ستقع هذه النقطة بعيداً عن الأحداثيات -1 و 1+ أى خارج مجال الرؤية و لذلك ستكون خارج نافذة العرض .

فمثلاً إذا كانت قيمة المحور Z تساوى 4 وحدات لذا ففى تلك الحالة سيكون مدى المجور Y هو [-4 , +4] داخل مجال رؤية الكاميرا . القيمة العظمى التى يأخذها المحور Y فى حال أن قيمة المحور Z تساوى 6 وحدات تقع داخل المدى [-6 , +6] ، و هذا الكلام ينطبق أيضاً على الرؤية من خلال المحور X و هكذا .....

لذا فلأى نقطة تقع داخل مجال رؤية الكاميرا و كان لها (X >-Z) أو (X < Z) أو (Y >-Z) أو (Y < Z) تكون تلك النقطة ظاهرة داخل نافذة الرؤية .

عندما نقوم بتحجيم المحورين X و Y لرأس ما لتتناسب مع المحور Z ، فى الواقع نحن نقوم بإنشاء نظام خيالى على شكل مخروط لنقوم بالرؤية من خلاله ، هذا النظام يزداد للخارج و بزاوية 90 درجة حول المحورين X و Y بمعنى 45 درجة لأعلى و 45 درجة لأسفل للمحور Y ، 45 درجة لليمين و 45 درجة لليسار للمحور X .

فى الصورة القادمة عندما يتقاطع الخطان الأحمران – عند مكان وجود الكاميرا – توجد فى هذه النقطة زاوية قائمة تسمى زاوية الرؤية – أى الزاوية التى نرى من خلالها الأشياء التى أمامنا . داخل المخروط الذى أمامنا أى نقط داخله تكون ظاهرة للكاميرا لإن محوريها X و Y يقعا داخل حدود نافذة العرض .

لذا فالكاميرا خاصتنا ستكون لها مجال رؤية بزاوية مقدارها ٩٠ درجة لإن المعادلات المشروحة سابقاً دائماً ما ستنتج قيم محصورة بين تلك الزاوية .

و بالرغم من عدم وجود المخروط ذو الزاوية ٩٠ درجة و أيضاً عدم وجود كاميرا داخل عالمنا الخيالى ، لذا فمن الجيد معرفة ما تقوم به تلك الدول لتحويل رؤوس المجسمات من منظور ثلاثى الأبعاد إلى منظور ثنائى الأبعاد . كل ما تقوم به هذه الدوال هو قسمة قيمة المحورين X و Y على المحور Z ، تلك العملية تتسبب فى تمدد أو فى إنكماش المجسم كلما أقترب أو أبتعد عن الكاميرا على التوالى . لذا و بالنظر إلى الصورة التالية نلاحظ أن مجموع المدى بين الخطين الأحمرين العلوى و السفلى للمخروط عند أى قيمة للمحور Z تقع دائماً فى المدى [1+ , 1-] .

و أيضاً فى تلك الصورة نلاحظ أنه كلما زادت Z كلما زادت المنطقة التى ستقع فى المدى [1+ , 1-] و الأشياء ستبدأ فى الإنكماش كلما أبتعدنا عن مركز نافذة العرض .

الصورة التالية تظهر مجموعة من النقاط رسمت على نفس مستوى المحور Y داخل مجال الرؤية ، نلاحظ أن كل نقطة أعلاها قيمة المجور Z لها كما نلاحظ إزدياد قيمة Z كلما تقدمنا للأمام .

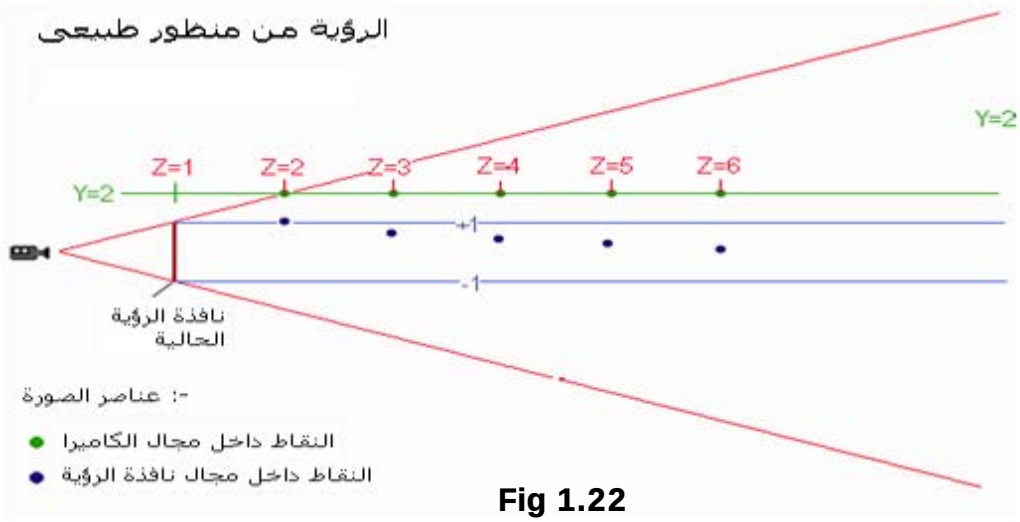


Fig 1.22

المعادلات سابقة الذكر تجعل الخطان الأحمران سنكمشوا حتى يصبحوا متوازيان و المسافة بينهما تصبح مساوية لمساحة نافذة العرض . الخطان الزرقاوان يظهروا شكل المحروط بعدما ينكمش ليصبح على شكل صندوق ، لذا فكلما زادت Z كلما أدت إلى زيادة نسبة الإنكماش .

فى الصورة السابقة ، توجد خمسة نقاط داخل مجال الرؤية – الدوائر الخضراء . كل منها له قيمة تبدأ من 2+ و تضاف إليها قيم المحور Z ، تأثير معادلات الرؤية سيظهر عندما تنظر للنقاط بعد تطبيق المعادلات عليها – النقاط الزرقاء . كل النقاط الزرقاء قد أنكمشت فى المدى [1+ , 1-] ، و بالرغم من أن النقاط الخضراء لها قيمة من المحور Y متماثلة داخل مجال الرؤية ، إلا أنه عند تطبيق معادلات الرؤية عليها أخذت كل نقطة قيمة مختلفة من المحور Y .

فى العديد من كتب الرياضيات ستجد أن معادلات الرؤية لها الشكل التالى :

$$Xp = \frac{X}{Z \mid d} \qquad Yp = \frac{Y}{Z \mid d}$$

المشكلة الموجودة بمعادلات الرؤية التى أستخدمناها من قبل تتمثل فى أنها **دائماً** ما تعطى إحداثيات بناءاً على زاوية رؤية مقدارها ٩٠ درجة ، و لكننا نفضل – كمبرمجين او كلاعبين – أستخدام أسلوب الرؤية الخرة لنسمح بالتحكم الكامل فى كيفية عرض المحسمات أمام الكاميرا ، و من أجل تحقيق ذلك ، نقوم بإضافة متغير جديد للمعادلة و هو يمثل بالحرف (d). هذا الإسلوب يتيح لنا تحريك نافذة العرض بعيداً أو قريباً من الكاميرا ، و حيث أن حجم نافذة العرض ثابت دائماً لا يتغير و يقع فى المدى [1+ , 1-] ، لذا فتحريك نافذة العرض بعيداً عن الكاميرا يجعل المخروط يصغر ، و كذلك عند تحريك المخروط ليقترب من الكاميرا فإن حجمه يكبر . المعادلات الجديدة تتيح لنا تغيير مجال الرؤية بنفس الإسلوب الذى يتبعه المصور عند التصوير فهو يقوم بتصغير العدسة أو تكبيرها حسب المشهد الذى يريد تصويره .

الصورة القادمة ترينا كيف أن تحريك نافذة العرض قد يؤثر على مجال الرؤية :

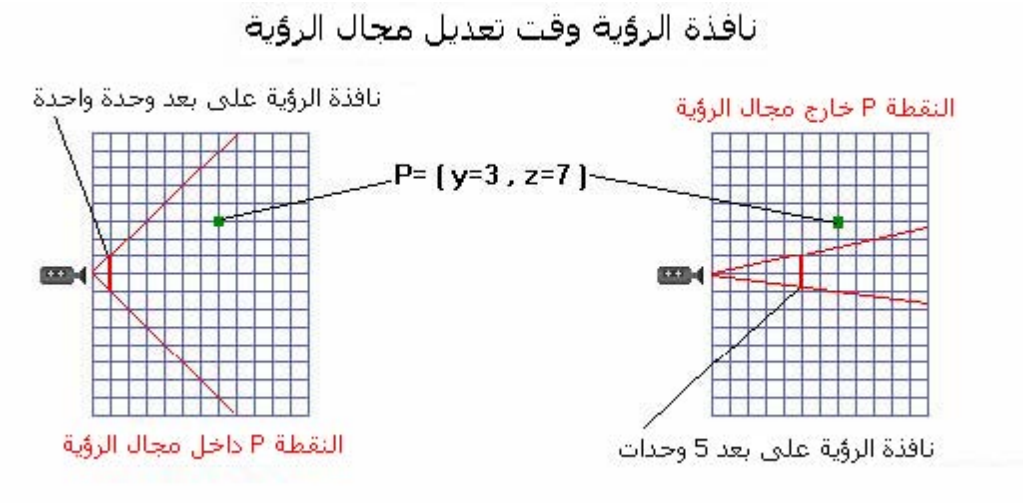


Fig 1.23

فى الصورة السابقة المخروط يبدو صغير جداً عندما تكون نافذة العرض على بعد ٥ وحدات من من الكاميرا ، ويبدو أكبر عندما تكون المسافة بينه و بين الكاميرا وحدة واحدة ، المسافة بين نافذة العرض و الكاميرا تمثل بالحرف d وهى أختصار للكلمة distance .

و حيث أن هذا الإسلوب يعمل جيداً ، لذا ففى محرك الإكساء الخاص بنا سيتم وضع نافذة العرض على مسافة ثابتة من الكاميرا و هى وحدة واخدة ، و لذلك عند تطبيق المعادلات الجديدة السابقة سنجد أن النتيجة هى نفس معادلاتنا القديمة كالتالى :

$$Xp = \frac{X}{Z \mid 1} = \frac{X}{Z}$$

$$Yp = \frac{Y}{Z \mid 1} = \frac{Y}{Z}$$

و مع ذلك من الممكن أن نحدث التأثير الذى نريده و الذى يحدث سابقاً بإستخدام المتغير d و لكن بإسلوب مختلف ، داخل الكود الخاص بنا و بمساعدة الـ DirectX سنستمر بإستخدام أسلوب الرؤية بزاوية ٩٠ درجة و لكن هذا سيتم خلف الستار و بعد ذلك سنقوم بإستخدام مصفوفة الرؤية Projection Matrix لتغيير شكل مجسماتنا قبل القسمة على المحور Z لإحداث تأثير الزؤية الحرة ، سنقوم بمعرفة المزيد عن مصفوفة الرؤية لاحقاً فى الدرس القادم ، و لذلك و حتى الأن سنقوم بإستخدام إسلوب الرؤية بزاوية ٩٠ درجة .

الرسم داخل إحداثيات الشاشة Screen Space Mapping

الخطوة الأخيرة الأن هى معرفة إحداثيات الشاشة لنقوم برسم المجسمات داخلها . و هذا سيتطلب منا تحويل الإحداثيات الخاصة برؤوس الكائن من إحداثيات نافذة الرؤية وهى [1+ , 1-] إلى إحداثيات الشاشة و هى طول و عرض الشاشة و المعادلات قد تبدو كذلك :

ScreenX= (ProjVertex.x * ScreenWidth) / 2 + ScreenWidth / 2

ScreenY= (-ProjVertex.y * ScreenHeight) / 2 + ScreenHeight / 2

فالنفترض أن الشاشة التى سيتم العرض عليها أبعادها 640x480 و لدينا رأس تم رسمها داخل نافذة الرؤية فى الإحداثى (0 , 0) ، لذا فهذا يعنى أنها سترسم فى منتصف الشاشة :

ScreenX= (0 * 640) / 2 + 640 / 2

ScreenX= 0 + 320

ScreenX= 320

مثال أخر و سيكون به المحور X يساوى 1- داخل نافذة الرؤية ، هذا سيجعل الرأس موجوده فى أقصى الجانب الأيسر لنافذة العرض (و كذلك أيضاً بالنسبة للشاشة) :

ScreenX= (-1 * 640) / 2 + 640 / 2

ScreenX= -320 + 320

ScreenX= 0

بالنسبة للمحور Y يتم تحويل إحداثياته إلى أحداثيات الشاشة بنفس الإسلوب السابق مع فارق واحد و هو أنه داخل مجال الرؤية المحور Y يأخذ قيمة موجبة كلما أرتفع لأعلى و يأخذ قيمة سالبة كلما إنخفض لأسفل . و هذا **معاكس** لنظام إحداثيات الشاشة حيث يكون المحور Y ذو قيمة صفر فى أعلى الشاشة و يزيد بقيمة موجبة كلما أتجهنا لأسفل ، و لذلك سنحتاج لأن نجعل قيمة المحور Y للرأس عند تحويلها لإحداثيات الشاشة أن تأخذ **أشارة عكس الإشارة الخاصة بها** أى **نقوم بضرب قيمة المحور Y فى الرقم 1-** .

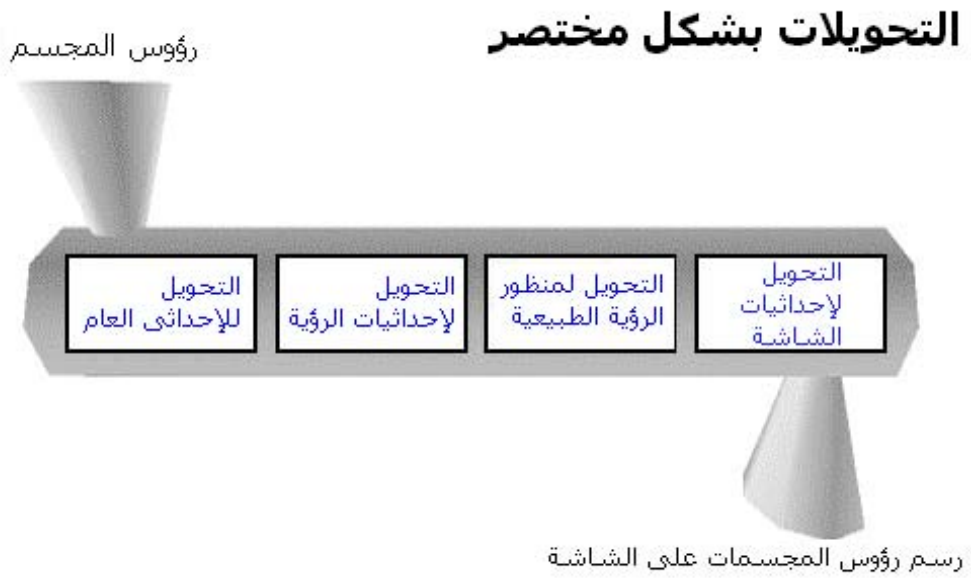
المثال التالى بأتى ليجعلنا نتأكد من صحة معادلتنا ، فالنفترض أن Y تساوى 1 داخل نافذة الرؤية ، و هذا يعنى أن الرأس موجوه فى أعلى نافذة الرؤية (و لابد ان تكون كذلك بالنسبة للشاشة) :

ScreenY= **-** (1 * 480) / 2 + 480 /2

ScreenY= -240 + 240

ScreenY=0

بمجرد أن تكون كل رؤوس المجسمات داخل نظام إحداثيات الشاشة ، نكون بذلك جاهزين لرسم خطوط بينهم ، و الناتج هو عبارة عن مجسم مفرغ للشكل النهائى لمجسمنا .



الكود القادم هو عبارة عن كود منطقى مطور للدالة DrawPrimitive ، فى المثال التالى نقوم بتمرير مؤشر للكائن الذى سيتم رسم أسطحه و أيضاً تمرير مؤشر للسطح المراد رسمه ، إحتياجنا لمؤشر الكائن يتمثل فى معرفة معلومات حول إحداثياته و أيضاً معلومات عن وضعه – إحداثيات الدوران الخاصه به حتى يتم تحويل السطح المراد رسمه إلى أحداثيات اللعبة World Space . نقوم أسضاً بتمرير مؤشر للكاميرا حتى نستطيع معرفة معلومات حول مكانها و إحداثيات دورانها حتى نستطيع تطبيق معادلات الرؤية عليها قبل رسمها داخل نافذة الرؤية و من ثم إلى شاشة العرض النهائية .

```
void DrawPrimitive( CObject *pObject , CPolygon *pPoly , CCamera *pCamera )
{
	CVertex CurrVertex;
	CVertex PrevVertex;

	// Retrieve object angles;
	float Opitch = pObject->RotationX;
	float Oyaw = pObject->RotationY;
	float Oroll = pObject->RotationZ;

	// Retrieve Camera angles
	float Cpitch = pCamera->RotationX;
	float Cyaw = pCamera->RotationY;
	float Croll = pCamera->RotationZ;

	// Loop round each vertex transforming as we go
	for ( USHORT v = 0; v < pPoly->m_nVertexCount + 1; v++ )
	{
		// Store the current vertex
		CurrVertex = pPoly->m_pVertex[ v % pPoly->m_nVertexCount ];

		// WORLD SPACE TRANSFORMATION
		// Apply any object rotations if applicable

		if (Opitch) // rotate object about its x axis like pitching up and down
		{
			currVertex.y = currVertex.y * cos(Opitch) –
					currVertex.z * sin(Opitch);

			currVertex.z = currVertex.y * sin(Opitch) +
					currVertex.z * cos(Opitch);
		} // End if Pitch


		if (Oyaw) // rotate object about its Y axis like yawing left/right
		{
			currVertex.x = currVertex.x * cos(Oyaw) +
					currVertex.z * sin(Oyaw);

			currVertex.z = currVertex.x * -sin(Oyaw) +
					currVertex.z * cos(Oyaw);
		} // End if Yaw

		if (Oroll) // rotate object about its Z axis like rolling left or right
		{
			currVertex.x = currVertex.x * cos(Oroll) +
					currVertex.y * sin(Oroll);

			currVertex.y = currVertex.x * sin(Oroll) +
					currVertex.y * cos(Oroll);
		} // End if Roll


		// Now move the vertex into its world space position
		currVertex.x += pObject.PositionX;
```

```
currVertex.y += pObject.PositionY;
currVertex.z += pObject.PositionZ;

// VIEW SPACE TRANSFORMATION
// subtract the camera position from the vertex so its position is
// relative to the camera with the camera at the origin

currVertex.x -= pCam->PositionX;
currVertex.y -= pCam->PositionY;
currVertex.z -= pCam->PositionZ;

// if the camera is rotated, rotate the world the opposite way
// but the only difference
// from the object rotation is the negated parameter

if (Cpitch) // rotate camera about its x axis like pitching up and down
{
        currVertex.y = currVertex.y * cos(-Cpitch) –
                       currVertex.z * sin(-Cpitch);

        currVertex.z = currVertex.y * sin(-Cpitch) +
                       currVertex.z * cos(-Cpitch);
} // End if Pitch

if (Cyaw) // rotate cam around its Y axis
{
        currVertex.x = currVertex.x * cos(-Cyaw) +
                       currVertex.z * sin(-Cyaw);

        currVertex.z = currVertex.x * -sin(-Cyaw) +
                       currVertex.z * cos(-Cyaw);
} // End if Yaw

if (Croll) // rotate camera about its Z axis like rolling left or right
{
        currVertex.x = currVertex.x * cos(-Croll) +
                       currVertex.y * sin(-Croll);

        currVertex.y = currVertex.x * sin(-Croll) +
                       currVertex.y * cos(-Croll);
} // End if Roll

// PERSPECTIVE PROJECTION TRANSFORMATION
// divide x and y by z to project point onto 2D projection
// window in the –1 to +1 range

currVertex.x /= currVertex.z;
currVertex.y /= currVertex.z;



// SCREEN SPACE TRANSFORMATION
// Convert to screen space coordinates

vtxCurrent.x = vtxCurrent.x * SCREENWIDTH / 2 +
```

```
                    SCREENWIDTH / 2;

            vtxCurrent.y = -vtxCurrent.y * SCREENHEIGHT / 2 +
                    SCREENHEIGHT / 2;

            // If this is the first vertex, continue. This is the first
            // point of our first line.

            if ( v == 0 )
            {
                    vtxPrevious = vtxCurrent;
                    continue;
            }

            // Draw the line between this one and the previous vertex in the loop
            DrawLine( vtxPrevious, vtxCurrent, 0 );

            // Store this as new line's first point
            vtxPrevious = vtxCurrent;

        } // Next Vertex
}
```

بعدما يتم إستدعاء الدالة السابقة لكل وجه داخل كل كائن موجود داخل العلم التخيلى، سنجد امامنا نسخة ثنائية الابعاد لمجسماتنا الثلاثية الأبعاد من خلا كاميراتنا الوهمية .

ملحق الكلمات

الكلمة :	Game Engine
معناها :	محرك اللعبه
الشرح :	هو مجموعة من الملفات التى تستخدم فى التحكم فى مسار تشغيل اللعبه
الكلمة :	Rendering Engine ، Renderer
معناها :	محرك الإكساء
الشرح :	يستخدم فى إكساء المجسمات و إظهارها على الشاشة
الكلمة :	DirectX API
معناها :	----------
الشرح :	هى مجموعة من الـ Classes المستخدمة فى برمجة الـ Graphics و الـ Multimedia
الكلمة :	Maya , 3D Studio Max
معناها :	----------
الشرح :	أسماء بعض البرامج المستخدمة فى صناعة الـ 3D
الكلمة :	Meshes , Models , Objects , 3D Models
معناها :	مجسمات ثلاثية الأبعاد
الشرح :	و هى مثل أى شئ .. سيارة ، كوكب ، طائرة ، إنسان ، حيوان ..... و غيرهم
الكلمة :	Face , Polygons
معناها :	سطح أو وجه
الشرح :	و هو مجموعة من النقاط – رؤوس – وصلت معاً لتكون وجه ، و مجموعة من الأسطح تكون مجسم Mesh
الكلمة :	Texture Maps
معناها :	المكسيات
الشرح :	هى مجموعة من الصور التى توضع على المجسمات الـ 3D لتجعلها تعبر عن شكل معين.
الكلمة :	Geometry
معناها :	شكل هندسى تم إنشائه رياضياً
الشرح :	هو أحد الأشكال الهندسية البسيطة مثل مكعب ، أنبوب ، براد ..... إلخ
الكلمة :	Coordinate Systems
معناها :	نظم الإحداثيات
الشرح :	هى إحدى خطوط الأعداد المستخدمة فى وصف العلاقات فى الفراغ
الكلمة :	Origin
معناها :	نقطة الأصل أو مركز الإحداثيات
الشرح :	هى النقطة التى تتقاطع عندها خطوط الأعداد المستخدمة فى نظم الإحداثيات و موقعها دائماً يكون (0 , 0) فى الأنظمة ثنائية الأبعاد
الكلمة :	Screen Coordinate System
معناها :	نظام إحداثيات الشاشة
الشرح :	هو النظام المستخدم فى الرسم داخل شاشات الكمبيوتر و فيه لا وجود للأرقم السالبة

الكلمة :	Pixels
معناها :	---------
الشرح :	هى نقطة على الشاشة و لها خصائص مثل اللون و الشفافيه
الكلمة :	2D Cartesian Coordinate System
معناها :	نظام الإحداثيات الديكارتى للأشكال ثنائية الأبعاد
الشرح :	و هو نظام رياضى يتسخدم لتمثيل البيانات على خطين من خطوط الأعداد
الكلمة :	3D Cartesian Coordinate System
معناها :	نظام الإحداثيات الديكارتى للأشكال ثلاثية الأبعاد
الشرح :	و هو نظام رياضى يتسخدم لتمثيل البيانات على 3 خطوط من خطوط الأعداد
الكلمة :	Z , Y , X
معناها :	----------
الشرح :	هى رموز تمثل خطوط الاعداد المستخدمة فى نظم الإحداثيات
الكلمة :	Axis
معناها :	محور
الشرح :	و هو أسم يطلق على خط من خطوط الأعداد مثل محور الـ Y
الكلمة :	Left-Handed System
معناها :	قاعدة اليد اليسرى
الشرح :	و هى أحد القواعد المستخدمة فى نظم الإحداثيات الـ 3D و هى تفترض أن محور الـ Z يزيد للأمام – أى كلما نظرنا للأمام
الكلمة :	Right-Handed System
معناها :	قاعدة اليد اليمنى
الشرح :	و هى أحد القواعد المستخدمة فى نظم الإحداثيات الـ 3D و هى تفترض أن محور الـ Z يزيد للخلف
الكلمة :	OpenGL
معناها :	---------
الشرح :	أحد المكتبات المستخدمة فى برمجة الـ Graphics
الكلمة :	Object Local Space , Model Space
معناها :	فضاء المجسم الخاص
الشرح :	و هو مكان حيث يكون نقطة الأصل للمجسم حيث تتقاطع جميع أوتاره – مركز المجسم
الكلمة :	3D Point , Vertex
معناها :	رأس
الشرح :	هى مجموعة متكاملة من البيانات تصف نقطة واحدة داخل نظام 3D
الكلمة :	Back Face Culling
معناها :	إخفاء السطح الخلفى
الشرح :	هو أسلوب برمجي يقوم على عدم رسم الأسطح الموجوده خلف المستخدم
الكلمة :	Winding Order
معناها :	ترتيب الرؤوس على الشاشة
الشرح :	هو رسم الرؤوس بإسلوب مرتب حتى يمكن رسم خط بين كل نقطتتين متقابلتين و متتاليتين على سطح واحد

الكلمة : Polygon Winding Order معناها : ترتيب الرؤوس على الأسطح الشرح : هو ترتيب الرؤوس فى إتجاه عقارب الساعة
الكلمة : Translation معناها : نقل الإحداثيات الشرح : هو إضافة إحداثى معين إلى كل رؤوس سطح معين حتى يتم نقله من مكانه إلى مكان أخر و هى عمليه تتم فقط على الرؤس
الكلمة : Transformation معناها : تحويل الإحداثيات الشرح : هى عملية تحريك مجسم من مكان إلى أخر بعد نقل رؤوسه و عملية تحويل الإحداثيات تتم فقط على الأسطح و المجسمات
الكلمة : World Space معناها : المجال العام او فضاء العالم التخيلى أو فضاء اللعبه الشرح : و هو المكان الذى تتواجد داخله جميع عناصر اللعبة من مجسمات .... إلخ
الكلمة : Instancing معناها : ---------- الشرح : هى عملية إنشاء اكثر من كائن من نفس النوع كأن تنشئ 3 كائنات و نوعهم CVertex
الكلمة : Frame معناها : --------- الشرح : و يمثل مرحلة من مراحل تحرك كائن ما داخل فضاء العالم التخيلى
الكلمة : Animation معناها : حركة الشرح : هى عملية تحرك كائن معين من مكان إلى أخر
الكلمة : Rotation معناها : تدوير الشرح : هو عملية يتم فيها دوران كائن ما حول شئ ما – نقطة الأصل مثلاً
الكلمة : Degree معناها : درجة الشرح : هو نظام قياسى يتم فيه قياس الزوايا بالدرجات
الكلمة : Radian معناها : -------- الشرح : هو نظام يتم التعبير عن زاوية معينة برقم عشرى
الكلمة : Camera معناها : كاميرا الشرح : هى كائن برمجي يتيح للاعب رؤية ما يواجه الكاميرا
الكلمة : Camera Space , View Space , Field Of View (FOV) معناها : نظام الكاميرات ، منطقة الرؤية ، مجال الرؤية الشرح : هو إمكانية رؤية ما تواجهه الكاميرا

الكلمة : Pitch معناها : ------- الشرح : تمثل التحرك لأعلى و أسفل حول المحور X
الكلمة : Yaw معناها : ------- الشرح : تمثل التحرك لليمين و اليسار حول المحور Y
الكلمة : Roll معناها : ------- الشرح : تمثل التحرك للأمام و الخلف حول المحور Z
الكلمة : Shrink معناها : تتقلص أو تصغر الشرح : هذا المصطلح يطلق على الكائنات عندما تقترب من مركز الرؤية
الكلمة : Grow معناها : تنمو أو تكبر الشرح : هذا المصطلح يطلق على الكائنات عندما تبتعد عن مركز الرؤية
الكلمة : Coordinate (Viewport Space , Projection Space , Clip-Space) معناها : نظام الرؤية المحدودة ، نظام الرؤية المختصر ، نظام عرض الصورة المتحركة الشرح : هو نظام إحداثى يستخدم للتعبير عن أحداثيات الرؤية داخل مجال الكاميرا عن طريق إحداثيات ثنائية الأبعاد
الكلمة : Projection Window معناها : نافذة الرؤية الحالية أو نافذة العرض الشرح : هى نافذة وهمية تستخدم لتحديد مجال الرؤية الذى سوف يتم عرضه بعد ذلك
الكلمة : d (in formula) معناها : ------- الشرح : هذا الحرف إختصار للكلمة (distance) و تعنى مسافة و تشير إلى المسافة بين نافذة العرض و الكاميرا و هذه المسافة هى التى تحدد مساحة الرؤية
الكلمة : Vector معناها : المتجه الشرح : هو كيان رياضى يصف مكان و طول نقطة معينة فى الفراغ
الكلمة : Vector Magnitude معناها : طول المتجه الشرح : هى عملية رياضية تهدف الحصول على طول متجه بإستخدام نظرية فيثاغورس
الكلمة : Collision Detection System معناها : نظام أكتشاف التصادمات الشرح : يستخدم هذا الإسلوب لمنع الكائنات من الإندماج فى حالة حدوث إصطدام بينهم
الكلمة : Scalar معناها : رقم , عدد الشرح : هو أى قيمة رقمية أياً كانت