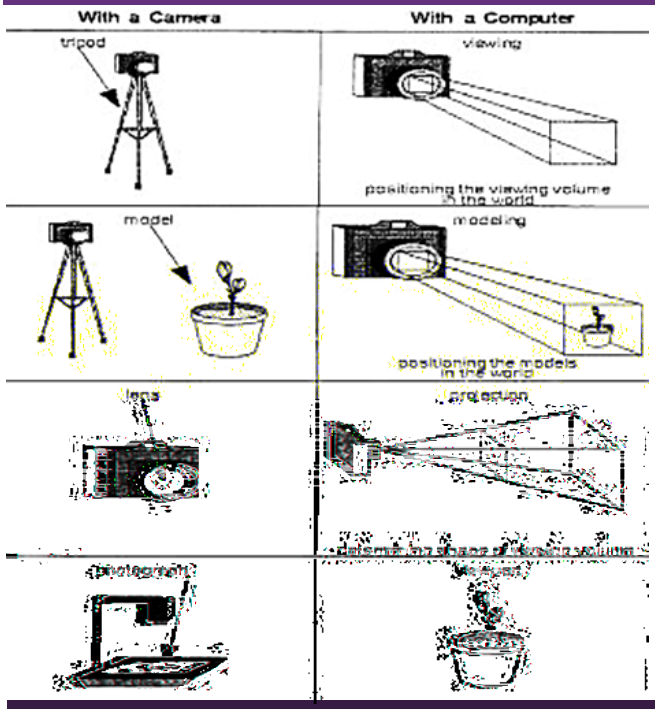
فريق الكريات

- يمكننا تشبيه عملية التحويل في الحاسب للحصول على المشهد بعملية التقاط صورة بالكاميرا فنقوم بـ:



1. تثبيت حامل الكاميرا وتوجيه الكاميرا إلى المشهد (viewing).
2. تنظيم المشهد ووضع عناصره في المكان المناسب (modeling).
3. اختيار عدسة الكاميرا وتعديل وضع التقريب والتبعيد zoom (projection).
4. تحديد حجم الصورة النهائية (viewport).
5. وبعد هذه الخطوات يمكننا التقاط الصورة بالكاميرا وبشكل مشابه رسم الصورة على الحاسب.

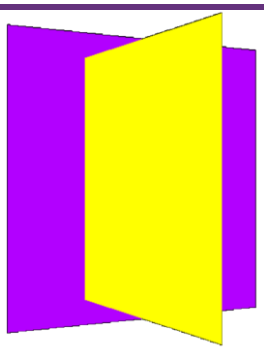
Depth Buffer

عندما يتم رسم كل بكسل يتم إسناد قيمة له تدعى z value، تمثل بعد النقطة المرسومة في هذا البكسل عن عين الناظر، وعندما نحتاج لرسم نقطة أخرى على نفس البكسل يتم فحص قيمة z value للبكسل مع قيمة الـ z للنقطة الجديدة فإذا كانت القيمة الجديدة أكبر (أقرب للناظر) يتم رسم النقطة وتغيير z value لهذا البكسل، وإلا يتم المحافظة على النقطة القديمة.

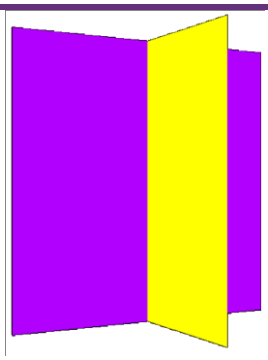
ملاحظة:

- يجب أن نكون قد هيئنا OpenGL لتحتوي على Depth Buffer من خلال PIXEL_FORMAT_DESCRIPTOR
- كما يجب تفعيل عملية اختبار هذا الـbuffer عن طريق: glEnable(GL_DEPTH_TEST)

- بدون Depth test



- مع Depth test



- ويمكننا حذف جميع القيم الموجودة في هذا الـ Buffer: glClear(GL_DEPTH_BUFFER_BIT)

تكون هذه الخاصة مفعلة تلقائياً by default في الـ OpenGL ولإلغاء تفعيلها نستخدم:
glDisable(GL_DEPTH_TEST)

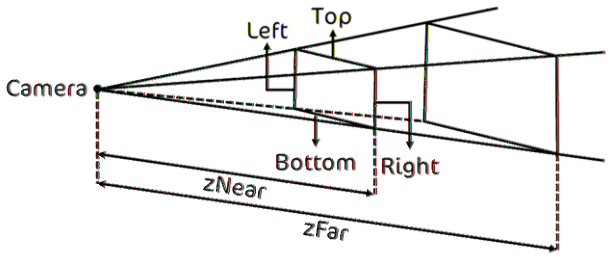
الإسقاط Projection

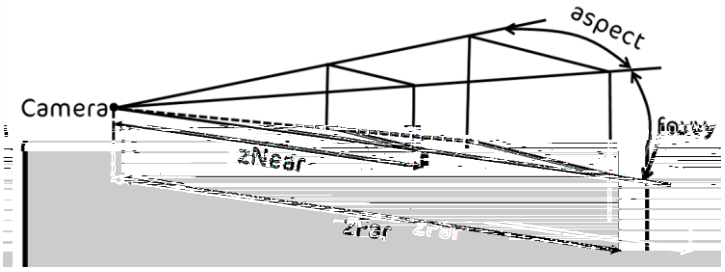
- تستخدم تحويلات الإسقاط لتحويل الرؤوس في المشهد، وقبل إصدار أي أمر تحويل إسقاط يجب تنفيذ التعليمة التالية glMatrixMode(GL_PROJECTION) وذلك لجعل تأثير التعليمات الحالية يطبق glLoadIdentity(); على مصفوفة الإسقاط وليس مصفوفة Model view.

يوجد نوعين من الإسقاط:

الإسقاط المنظوري (Perspective):

يتم رؤية الأجسام الواقعة داخل جذع هرم، وكلما كان الجسم أبعد عن الكاميرا كلما كان أصغر، يحدث ذلك لأن حجم الإظهار لهذا الإسقاط عبارة عن جزء من جذع الهرم.
يسبب تشوه الجسم بسبب تحويل جذع الهرم لمكعب، ولكنه أكثر واقعية

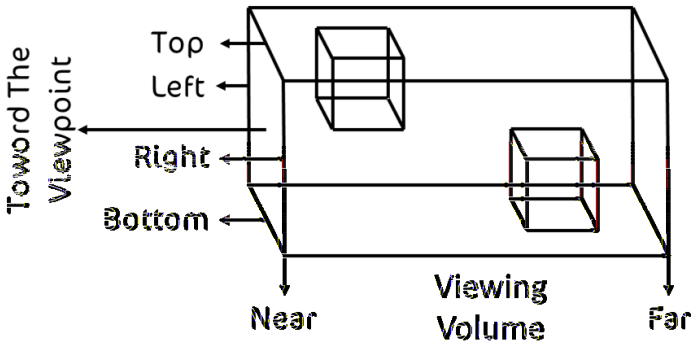





```
void gluPerspective ( GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar )
void glFrustum ( GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
    GLdouble znear, GLdouble zfar );
```

الإسقاط المتعامد (Orthographic):

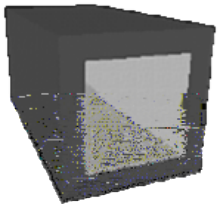
- يتم فيه رؤية الأجسام الواقعة داخل متوازي مستطيلات
- التابع المستخدم في هذا النوع من الإسقاط هو glOrtho( ) وله الشكل:



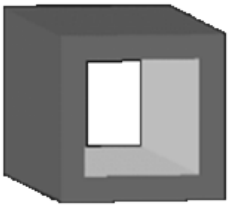

```
void glOrtho ( GLdouble left, GLdouble right,
    GLdouble bottom, GLdouble top,
    GLdouble near, GLdouble far );
```

- لاحظ الأشكال التالية:

Perspective



Orthographic



View port

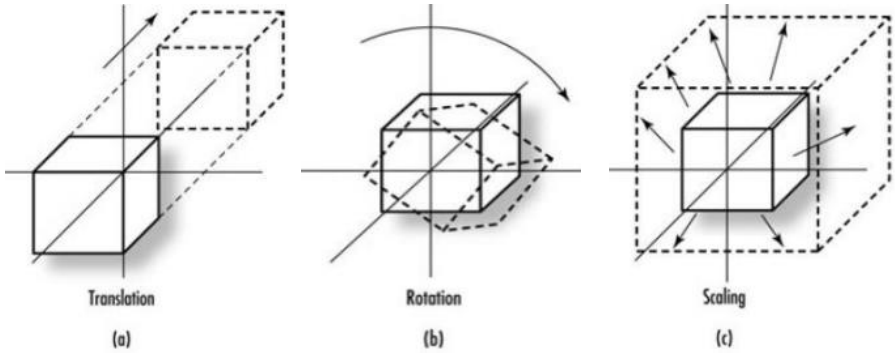
- هي المساحة التي يتم الاظهار عليها على الشاشة، يتحدد بإحداثيات مع طول وعرض:

glViewPort (x,y,width,height);

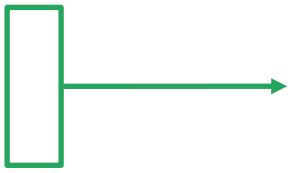
Modeling Transformations

لتحريك الكائن المراد رسمه لدينا 3 حركات:

1. الانسحاب (Translation)
2. التدوير (Rotation)
3. تغيير الحجم (Scaling)



في الـ OpenGL يتم تمثيل معلومات مركز الإحداثيات بواسطة المصفوفة الواحدية التي أبعادها وإن تغير قيمة أي عنصر فيها يؤدي إلى تغير بمركز الإحداثيات.

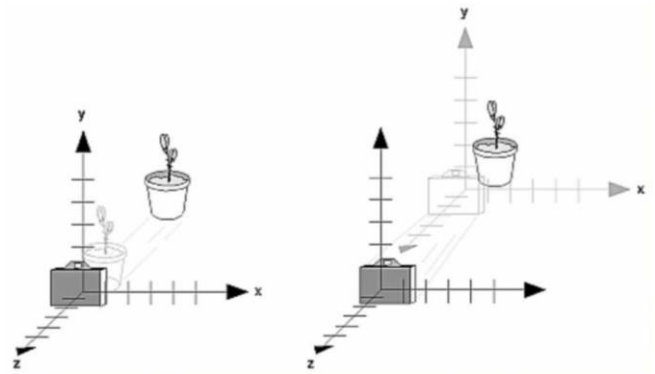


Center position
تمثل موضع مركز الإحداثيات وفي الحالة الابتدائية تكون (0,0,0)

أي عند الرسم على الشاشة يكون مركز الإحداثيات في منتصف الشاشة ولتغيير موضعه نغير في عمود الـ Position.
عند رسم نقطة ما يتم إعطاؤها موقع (x,y,z) بالنسبة لمبدأ الاحداثيات ولا علاقة له بموقع الكاميرا.

1. الانسحاب Translate:

هي عملية نقل الشكل من موقع لموقع آخر على مستوى الإحداثيات في أحد الاتجاهات الثلاثة.

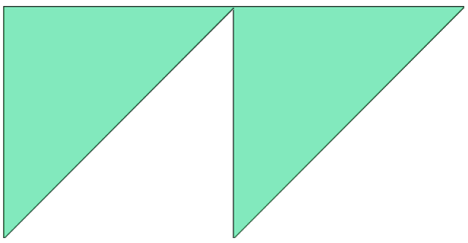


- لكن فعلياً كيف سننقل الشكل؟

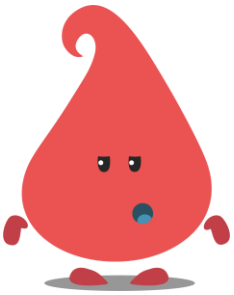
دعنا نفكر قليلاً: في حال قمنا برسم مثلث وأردنا تغيير مكانه ما سنفعله تلقائياً هو أننا سنغير الإحداثيات، لكن ماذا لو كان الشكل أعقد مكعب مثلاً هل سنغير إحداثيات جميع نقاطه، الأمر مكلف وصعب، لكن إذا نقلنا مركز الإحداثيات أي (نقطة واحدة) الأمر سيكون أسهل وسنحصل على النتيجة المطلوبة لأن الشكل سينتقل مع مركز الإحداثيات.

رياضياً: الانسحاب هو عملية ضرب بالمصفوفة التالية:

مثال:



```
glBegin(GL_TRIANGLES);
glVertex2d(0,  0);
glVertex2d(-1,  0);
glVertex2d(-1,  -1);
glEnd();
glTranslated(1,0,0);
glBegin(GL_TRIANGLES);
glVertex2d(0,  0);
glVertex2d(-1,  0);
glVertex2d(-1,  -1);
glEnd();
```



- نلاحظ في المثال السابق:

1. أن إحداثيات المثلث لم تتغير إنما تغير فقط مركز الإحداثيات والدليل على ذلك أن المثلث ارتسم بجانب المثلث القديم أي ارتسم بالنسبة للمركز الجديد وإلا لارتسم فوق المثلث القديم.
2. ونلاحظ أيضاً أن عملية الرسم تتم بعد نقل المركز؛ لأنه لا يمكن تغيير أو تعديل الشكل بعد رسمه.

Animation

في الرسوم المتحركة يتم عرض صور بشكل سريع جداً فيتوهم لنا أنها تتحرك فعلياً، وكذلك الأمر في الـ OpenGL حيث لجعل الشكل في حالة حركة مستمرة يتم استدعاء التابع DrawGLScene عند كل frame لتشغيل البرنامج بشكل مشابه لحلقة تكرارية.

- Frame: هي الصورة المرسومة التي تتغير 30 مرة في الثانية الواحدة (في النسخ الأحدث يمكن أن يكون العدد أكبر).

مثال:

في الكود التالي سيتحرك الشكل بقيمة X بالاتجاه الموجب لمحور الـX عند كل استدعاء لتابع لـ DrawGLScene، وفي كل استدعاء سيظهر المربع بموقع جديد؛ لأننا نزيد من قيمة الانسحاب X وبالتالي سيبدو الشكل بحالة حركة.

```
glTranslated(X,0,0);
Draw_Square();
X+=0.01;
```

ملاحظات:

نجد في بداية تابع الـ DrawGLScene() التعليمات الأساسية التالية:

- glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT):

حيث تقوم هذه التعليمة بمسح الذواكر المؤقتة الـ Buffer الخاصة باللون والعمق للشكل المرسوم وبما أن هذا التابع يستخدم 30 مرة في الثانية يتم مسح الـ Buffer بعد كل عملية رسم.

- تابع glLoadIdentity( ):

في كل مرة يعيد تحميل المصفوفة الواحدية وهذا يعيد مركز الإحداثيات إلى مركز الشاشة، أي يعود إلى الحالة الابتدائية بعد كل عملية رسم للتابع DrawGLScene( )

2. الدوران Rotate:

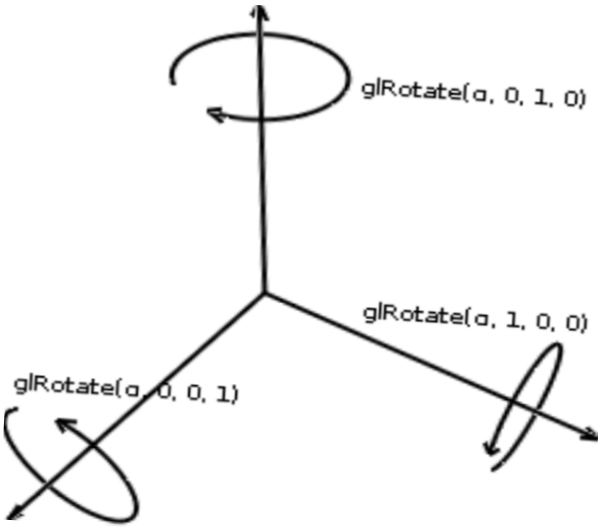
هذا التحويل يعمل على تدوير كل شكل هندسي وفق محور ويتم ذلك بواسطة التابع:

glRotateT(angle, X, Y, Z)

وT يمكن أن تكون f: float، d: double كما في الـ Translate

و angle زاوية الدوران، (X,Y,Z) هي إحداثيات النقطة التي تحدد محور الدوران.

- نحن نعلم لتدوير شكل ما يجب تحديد: المحور و زاوية الدوران



المحور: هو مستقيم أي يحتاج لنقطتين لكن في التابع السابق نحن نمرر له نقطة واحدة؟!

ذلك لأنه تلقائياً يأخذ النقطة الأولى مركز للإحداثيات والنقطة الثانية هي التي نمررها فيحدد المحور.

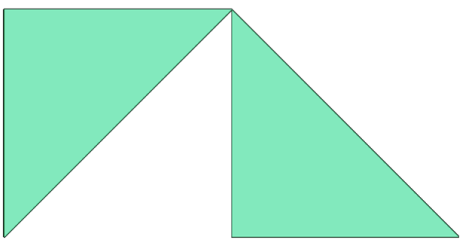
وبشكل عام نحن غالباً نتعامل مع المحاور X,Y,Z فقط.

ملاحظة: تقاس الزاوية بالراديان وإذا كانت 0 لا يحدث دوران

- عملية الدوران هي عملية ضرب بالمصفوفة ويوجد 3 مصفوفات كل مصفوفة خاصة لمحور:

مثال:

```
glBegin(GL_TRIANGLES);
glVertex2d(0,  0);
glVertex2d(-1,  0);
glVertex2d(-1,  -1);
glEnd( );
glRotated(90,0,0,1);
glBegin(GL_TRIANGLES);
glVertex2d(0,  0);
glVertex2d(-1,  0);
glVertex2d(-1,  -1);
glEnd( );
```



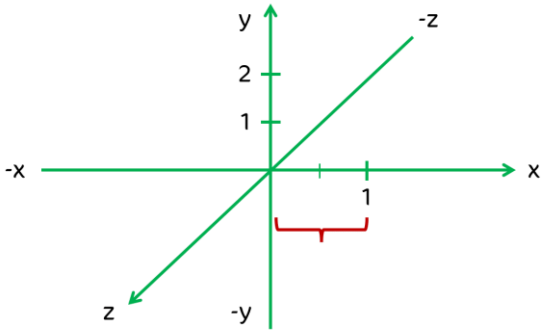
لاحظ أن الاتجاه الموجب للدوران هو عكس عقارب الساعة.

3. تغيير الحجم Scaling:

يقوم هذا التحويل بتغيير حجم أي شكل هندسي وفق محور أو أكثر ونستخدم لذلك التابع:

glScaleT(X, Y, Z)

- يقوم التابع بضغط الشكل أو تمديده وهو يقابل الضرب مع المصفوفة.
- هذا التابع كما في الدوران والانسحاب لا يغير من إحداثيات الشكل، وإنما يغير طويلة شعاع الواحدة الخاصة بمركز الإحداثيات، فالعوامل الافتراضية لقياس الإحداثيات هي (1,1,1) عند تطبيق عملية Scale ستتغير عوامل القياس.
- توضيح: Scale(2,1,1)
- المصفوفة التي تمثل عملية الـ scale هي:



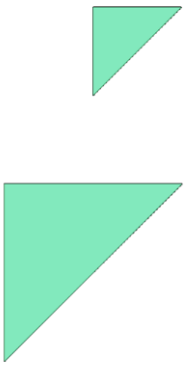
ملاحظات:

- يجب ألا يكون أحد معاملات هذا التابع صفر لأنه عندئذ سيختفي هذا المحور
- القيم أكبر من 1 للتكبير وأصغر من 1 للتصغير، والقيم السالبة تعني عكس الجسم على المحور.
- إذا كانت العوامل الثلاثة متساوية ستكون النتيجة شكل مشابه للشكل الأصلي ومتناسب معه من حيث أطوال الأضلاع والزوايا.

أمثلة:

.1

```
glBegin(GL_TRIANGLES);
glVertex2d(0, 0);
glVertex2d(-1, 0);
glVertex2d(-1, -1);
glEnd();
glTranslated(0,1,0);
glScaled(0.5,0.5,0.5);
glBegin(GL_TRIANGLES);
glVertex2d(0, 0);
glVertex2d(-1, 0);
glVertex2d(-1, -1);
glEnd();
```



.2

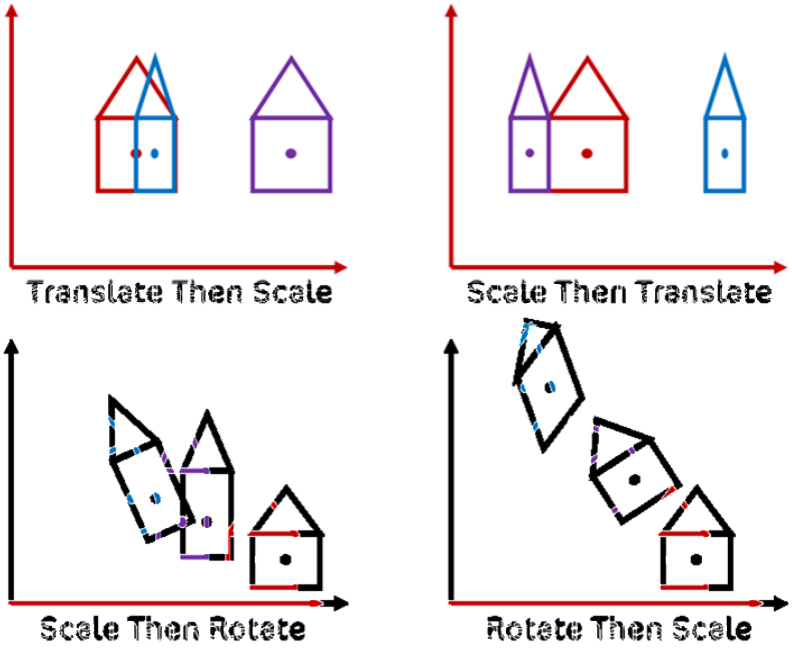
لاحظ كيف تحول شكل المربع إلى مستطيل.

```
glScaled(2,1,1);
glBegin(GL_QUADS);
glVertex3d(1,1,0);
glVertex3d(-1,1,0);
glVertex3d(-1,-1,0);
glVertex3d(1,-1,0);
glEnd();
```

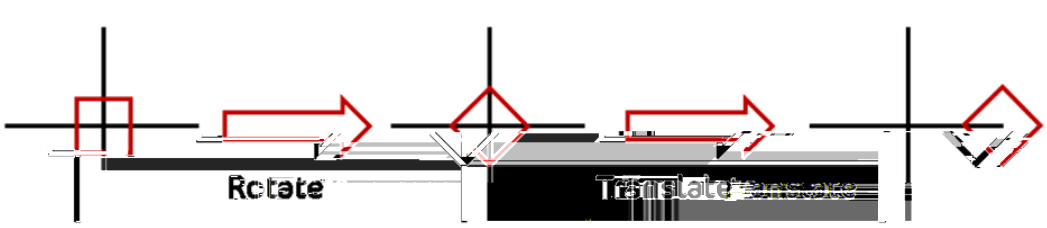
دمج التحويلات مع بعضها

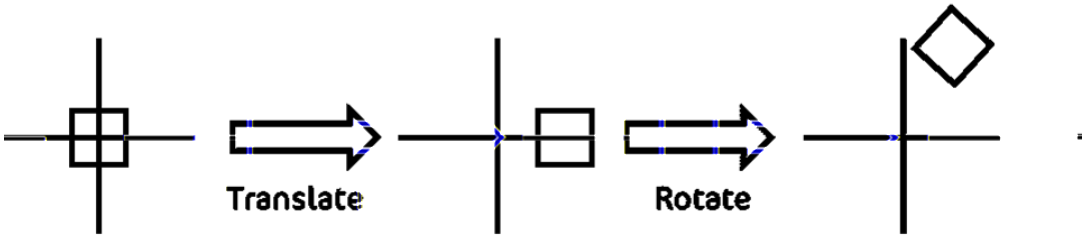
Modeling Transformation mixing them all

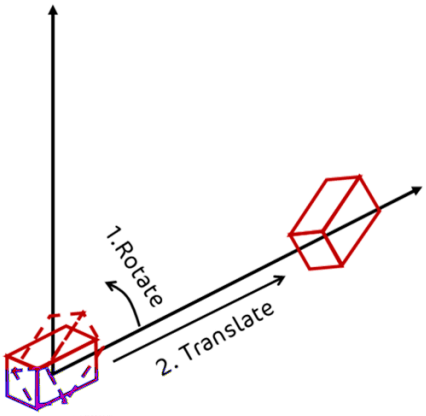
- إن كل تحويل يعادل ضرب مصفوفات وعملية ضرب المصفوفات ليست تبديلية فيمكننا القيام بالتحويلات معاً لكن اختلاف ترتيب هذه التحويلات يعطي نتائج مختلفة.

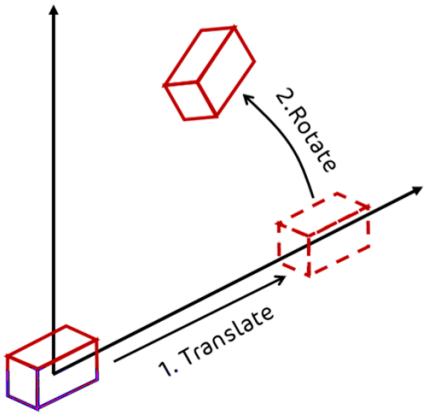


- الأحمر هو الشكل الأصلي
- البنفسجي هو المرحلة الانتقالية
- الأزرق هو النتيجة النهائية









تذكر:

في الـ OpenGL جميع التحويلات يتم تنفيذها على مركز الإحداثيات ومحاور الإحداثيات وليس على الشكل، وإن قلنا أن التدوير على الشكل فنحن نقصد ذلك مجازاً.

Push/Pop Matrix:

عند رسم عدة أشكال وتحريكها فإن مركز احداثيات الرسم سيتغير موضعه في كل مرة يتم التعديل على آخر مركز رسم وهذا قد يحدث أحياناً بعض الأخطاء في الرسم.

مثلاً إذا أردت عمل تحويل T وRg على جميع الأشكال التي سأرسمها ثم عمل تحويل Z على شكل واحد معين فقط وعدم تطبيقه على بقية الأشكال التي سأرسمها **فما الحل؟!**

هنا سنلجأ إلى ما يسمى بعملية الـ Push والـ Pop.

وفي الـ Push: نخزن المصفوفة التي أرسم عليها (مركز الرسم الحالي) حيث نكتب التابع قبل عملية الرسم glPushMatrix( );

وبعد عملية الرسم لنستعيد مركز الرسم الأصلي أو السابق لهذه الرسمة نستخدم التابع: glPopMatrix( );

وبهذه العملية يمكننا رسم أي شيء نريده ونقوم بأي تحويل نريده على الشكل دون التأثير على بقية الأشكال.

بمعنى آخر نحصر بين الـ glPushMatrix( ) والـ glPopMatrix( ) التعليمات التي نريدها أن تنفذ على الشكل المرسوم بداخلها فقط دون أن تؤثر على بقية الأشكال المرسومة في باقي الكود.

Viewing Transformation:

تسمح لنا هذه التحويلات بتغيير موقع وجهة نظر المراقب (المشاهد) ويشبه ذلك وضع الكاميرا وتوجيهها باتجاه الجسم، ويمكننا التحكم بالكاميرا من خلال التابع التالي:

```
void gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
               GLdouble centerX, GLdouble centerY, GLdouble centerZ,
               GLdouble upX, GLdouble upY, GLdouble upZ);
```

حيث:

eyeX, eyeY, eyeZ: تحدد موضع الكاميرا في المشهد (مركز عين الناظر) وفي الحالة الافتراضية تأخذ القيم (0,0,0)

centerX, centerY, centerZ: تحدد الاتجاه التي تنظر إليه الكاميرا (وجهة نظر العين) وقيمتها الافتراضية (0,0,-1)

upX, upY, upZ: يشير إلى المتجه العمودي على الكاميرا وقيمته الافتراضية (0,1,0) تظهر الفائدة من up في حال محاكاة حركة طائرة مثلاً في حال انقلاب طائرة ستبقى الإحداثيات ثابتة أما up ستتغير قيمته إلى (0,-1,0)

من المهم جداً فهم معاملات التابع gluLookAt من أجل التحريك في المشهد.

مثال:

نستخدم دالة الـ sin والـ cos في إحداثيات العين eyeY لإظهار الحركة على شكل موجة وهي تشبه حركة الإنسان الحقيقية أما centerVal فيجب أن تمثل نصف دائرة المحاكاة مدى نظر الإنسان، لأن الإنسان يستطيع أن يرى المجال نصف دائرة أمامه، أما الـ upVal فتبقى ثابتة.

انتهت المحاضرة