الكلية التطبيقية

بيانات الحاسوب

سنة رابعة

مدرس المقرر
عبدالرزاق المحمد المحمود

2026-2025

محاضرة (1) مقدمة إلى بيانات الحاسوب

أصبحت رسوم الحاسوب في العصر الحالي، جزءاً أساسياً من حياتنا اليومية، سواء في الألعاب الحاسوب، أو التصاميم الرسومية، أو الرسوم المتحركة، أو حتى في التطبيقات الطبية مثل التصوير بالرنين المغناطيسي.

تغطي هذه المحاضرة، المفاهيم الأساسية لهذا المجال، بدءاً من تاريخ نشوء رسوم الحاسوب، مروراً بالخوارزميات والتقنيات، وصولاً إلى التطبيقات العملية.

رسوم الحاسوب ليست مجرد رسم صور على شاشة، وإنما هي علم يجمع بين الرياضيات، الفيزياء، والبرمجة لإنتاج صور واقعية أو خيالية، على سبيل المثال، الرسوم المتحركة في الأفلام مثل فلم "أفاتار" أو "الأسد الملك" تستخدم تقنيات حاسوبية متقدمة لإنتاج مشاهد واقعية، كما أنها تستخدم في تصميم السيارات، حيث يمكن للمهندسين رؤية نموذج ثلاثي الأبعاد قبل صنع السيارة الحقيقة، وكذلك الأمر في تصميم الأبنية أو أعمال الديكور،،،

يعود تاريخ رسوم الحاسوب إلى ستينيات القرن الماضي، عندما طور إيفان ساذرلاند نظام Sketchpad في عام 1963، الذي كان أول برنامج تفاعلي للرسم على الحاسوب، ومنذ ذلك الحين، تطورت التقنيات مع ظهور معالجات الرسوم GPUs في التسعينيات، مما أدى إلى ثورة حقيقية في الألعاب والرسوم المتحركة.

الأساسيات في رسوم الحاسوب

يتم تمثيل الصور باستخدام وحدات أساسية تسمى البكسلات Pixels كل بكسل هو نقطة صغيرة على الشاشة تحمل قيمة لونية، على سبيل المثال، في شاشة بدقة 1920x 1080، هناك أكثر من مليوني بكسل.

هناك نوعان رئيسيان من الرسوم:

الرسوم النقطية Raster Graphics والرسوم الشعاعية Vector Graphics

تعتمد الرسوم النقطية على شبكة من البكسلات، كالصور الرقمية التي يتم معالجتها في برنامج الـ Photoshop بينما تعتمد الرسوم الشعاعية على معادلات رياضية لوصف الأشكال، مثل الخطوط والدوائر، وهذا النوع يستخدم في التصاميم التي تحتاج إلى تكبير دون فقدان الجودة، كما هو الأمر في برنامج الـ Illustrator .

بالنسبة لتمثيل الألوان فيتم استخدم نموذج Red, Green, Blue (RGB) لخلط الألوان الأساسية، حيث يتم تمثيل كل لون من المكونات الثلاث السابقة بقيمة من 0 إلى 255، على سبيل المثال، اللون الأحمر هو 255، 0، 0، هناك نموذج آخر CMYK وغالباً من يستخدم للطباعة، لكن RGB هو الأكثر شيوعاً بالنسبة لشاشات العرض.

أما بالنسبة للإحداثيات فيتم استخدام نظام الإحداثيات الكارتيزي، حيث يمثل X المحور الأفقي و Y المحور الرأسي أو العمودي وتكون النقطة 0,0 عادة في أعلى اليسار أو في أسفل اليسار حسب نظام العرض.

خوارزميات الرسم الأساسية

تعتبر خوارزميات الرسم من أهم الأمور المتعلقة برسوم الحاسوب، أهم هذه الخوارزميات:

1- خوارزمية رسم خط مستقيم: خوارزمية Bresenham، ترسم خطاً مستقيماً بين نقطتين x1, y1 و x2, y2، ولهذا الغرض تحسب الفرق dx = x2 – x1 ، dy = y2 – y1 ثم تبدأ من النقطة الأولى وتضيف خطوات بناءً على الخطأ المتراكم، تعتبر هذه الخوارزمية مناسبة للأجهزة ذات الموارد المحدودة.

2- خوارزمية رسم دائرة: خوارزمية Midpoint Circle Algorithm تبدأ من مركز الدائرة xc, yc وباستخدام نصف القطر r ، تقوم برسم نقاط في ثمانية أقسام متماثلة لتقليل الحسابات.

3- خوارزمية رسم المضلعات: خوارزمية Scanline Fill تقوم هذه الخوارزمية بمسح الشاشة أفقياً وملء البكسلات ضمن حدود المضلع.

الخوارزمية	الاستخدام	المزايا	العيوب
Bresenham Line	رسم الخطوط	سريعة، لا تحتاج عائمة	محدودة بالخطوط المستقيمة
Midpoint Circle	رسم الدوائر	كفاءة عالية في التماثل	تحتاج حسابات مربعة
Scanline Fill	ملء المضلعات	فعالة للمناطق الكبيرة	يزداد التعقيد في المضلعات المعقدة

التحويلات في رسوم الحاسوب

التحويلات هي عمليات رياضية لتغير موقع أو شكل الكائنات Objects.

التحويلات في الرسوم ثنائية الأبعاد هي: الإزاحة Translation ، التكبير/التصغير Scaling ، والدوران Rotation.

الإزاحة: $x' = x + tx$ ، $y' = y + ty$

التكبير: $x' = x * sx$ ، $y' = y * sy$

الدوران حول المحور: $x' = x * \cos\theta - y * \sin\theta$ ، $y' = x * \sin\theta + y * \cos\theta$

هذه التحويلات تمثل بمصفوفات، مما يسمح بتسلسلها (أي دمج عدة تحويلات معاً في عملية واحدة عن طريق ضرب المصفوفات الخاصة بها).

النمذجة ثلاثية الأبعاد

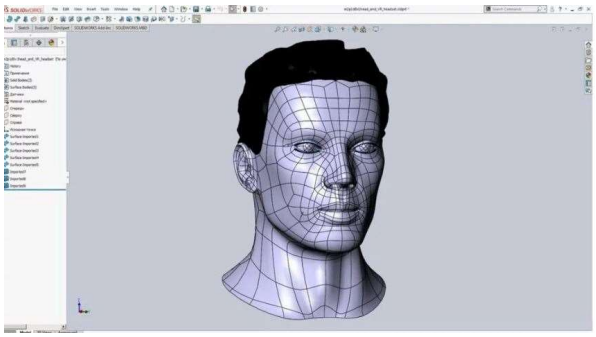
النمذجة ثلاثية الأبعاد 3D Modeling هي عملية إنشاء تمثيل رياضي لكائن ثلاثي الأبعاد باستخدام برمجيات متخصصة، لإنتاج صور واقعية في الألعاب، الأفلام، التصميم الصناعي، والتطبيقات الطبية.

بدأت النمذجة ثلاثية الأبعاد في الستينيات مع أنظمة مثل Sketchpad ، لكنها تطورت بشكل كبير مع انتشار الحواسيب الشخصية في التسعينيات.

لإنتاج الرسوم ثلاثية الأبعاد، نستخدم نماذج مثل: المضلعات Polygons أو المنحنيات Curves أو الـ Mesh الذي يتكون من رؤوس Vertices ، حواف Edges ، ووجوه Faces

ولعرض النموذج ثلاثي الأبعاد على شاشة ثنائية البعد نستخدم الإسقاط Projection ، مثل الإسقاط المنظوري Perspective Projection الذي يعطي شعوراً بالعمق.

أنواع النمذجة ثلاثية الأبعاد



1- النمذجة المضلعة: تعتمد على بناء النموذج من مضلعات مثل المثلثات أو المربعات، فيتم تقسيم السطح إلى شبكة من الرؤوس Vertices ، الحواف Edges ، والوجوه Faces ، وتستخدم هذه النمذجة في الألعاب لكفاءتها في الرسم السريع.

2- النمذجة المنحنية: وتستخدم لإنشاء أسطح ناعمة وهذا النوع مناسب للتصاميم الصناعية مثل السيارات، التي تحتاج إلى دقة عالية – دون تشوه – عند تكبير التصميم.

3- النمذجة بالتقسيم السطحي: تبدأ بنموذج بسيط ومن ثم تقسمه إلى أجزاء أصغر للحصول على سطح أكثر نعومة وتستخدم في الرسوم المتحركة.

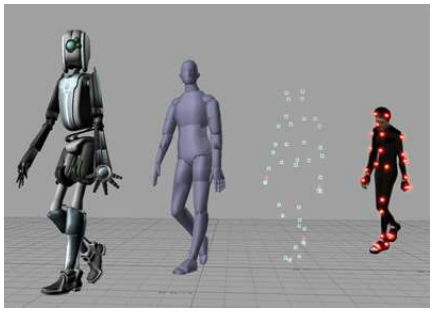
4- النمذجة الرقمية: وهي تشبه النحت التقليدي، يمكن تنفيذها من خلال برامج متخصصة مثل ZBrush .

5- النمذجة البارامترية: تعتمد على معادلات وبرامترات قابلة للتعديل، مثل برنامج الـ Autocad.

النوع	التطبيقات	المزايا	العيوب
النمذجة المضلعة	الألعاب	سريعة	سيئة إذا كانت المضلعات قليلة
النمذجة المنحنية	التصميم الصناعي	دقيقة وناعمة	معقدة في الحسابات
النمذجة بالتقسيم السطحي	الرسوم المتحركة	سهلة التحويل إلى ناعمة	تحتاج موارد حوسبة
النمذجة الرقمية	الفن الرقمي	مرنة، انطباع فني	غير دقيقة للتصاميم الهندسية
النمذجة البارامترية	التصاميم الهندسية	مرنة، ودقيقة	معقدة، وحجوم ملفاتها كبير

الأدوات:

Blender	يدعم جميع التقنيات
Maya	الرسوم المتحركة والأفلام
3ds Max	التصميم المعماري والألعاب
ZBrush	النحت الرقمي
Autocad	التصميم الهندسي



الإضاءة والتظليل Lighting & Shading

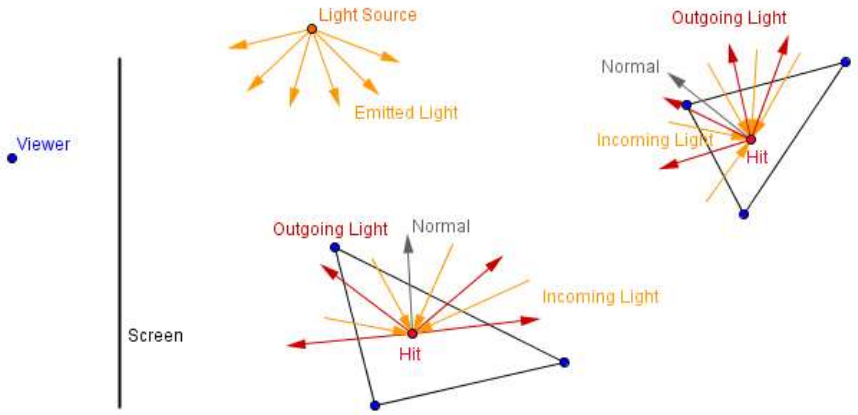
الإضاءة والتظليل هما عنصران أساسيان في رسوم الحاسوب ثلاثية الأبعاد، حيث يساهمان في جعل النماذج تبدو واقعية ومؤثرة، بدأت دراسة الإضاءة في السبعينيات مع نماذج مثل Phong, Lambertian ، وتطورت إلى تقنيات حديثة مثل Ray Tracing، وهذه النماذج هي جزء من عملية الـ Rendering، التي تقوم بحساب الإضاءة بعد النمذجة والتحويلات.

الإضاءة

تشير إلى كيفية تفاعل الضوء مع الأسطح.

التظليل

يحدد كيفية حساب وتوزيع الألوان والكثافة الضوئية عبر السطح.



أنواع الإضاءة في رسوم الحاسوب

تُقسم الإضاءة إلى أربعة أنواع رئيسية، بحسب طريقة تفاعل الضوء مع السطح:

1- الإضاءة المحيطية Ambient Lighting : تمثل الضوء غير المباشر الذي ينتشر في كل اتجاه، مثل الضوء المنعكس من الجدران في غرفة، لا تعتمد على اتجاه مصدر الضوء أو زاوية المشاهد.

وتُحسب من خلال العلاقة $L * K$ ، حيث: L كثافة الإضاءة المحيطية و K معامل الانعكاس المحيطي، هذا النوع يمنع المناطق المظلمة من أن تكون سوداء تماماً.

2- الإضاءة المنتشرة Diffuse Lighting تعتمد على نموذج Lambertian ، حيث ينعكس الضوء بشكل متساوٍ في جميع الاتجاهات بغض النظر عن زاوية المشاهد.

ويحسب من خلال المعادلة $L * K * \cos\theta$ ، حيث θ الزاوية بين متجه الضوء ومتجه السطح الطبيعي وهذا يعطي شعوراً بالعمق، كما في الأسطح غير اللامعة مثل الورق.

3- الإضاءة المرآتية Specular Lighting: تمثل الانعكاس اللامع، كما في المعادن أو الزجاج، وهذه تعتمد على زاوية المشاهد، وتُحسب من خلال العلاقة L * (K * cos α)^n ، حيث α الزاوية بين متجه الانعكاس ومتجه المشاهد، و n معامل يحدد شدة اللمعان.

4- الإضاءة الانبعاثية Emissive Lighting: الضوء المنبعث من الكائن نفسه، مثل مصباح أو شاشة.

النوع	الوصف	التأثير الرئيسي
الإضاءة المحيطية	ضوء غير مباشر	يملأ الظلال
الإضاءة المنتشرة	انعكاس متساوي	يعطي العمق
الإضاءة المرآتية	انعكاس لامع	يخلق لمعاناً
الإضاءة الانبعاثية	ضوء منبعث	يجعل الكائن مضيئاً

أبرز نماذج الإضاءة الرياضية هو نموذج Lambertian والذي يمثل الإضاءة المنتشرة ، بينما يجمع نموذج Phong بين الأنواع الثلاثة الأولى (المحيطية، المنتشرة، المرآتية).

طرق التظليل

يحدد التظليل كيفية تطبيق الإضاءة عبر السطح:

1- التظليل المسطح Flat يحسب الإضاءة مرة واحدة لكل مضلع، باستخدام متجه طبيعي واحد.

2- تظليل غورو Gouraud يحسب الإضاءة في الرؤوس Vertices ثم يمزج بينها عبر السطح.

3- تظليل فونغ Phong يمزج المتجهات الطبيعية عبر السطح، ثم يحسب الإضاءة لكل بكسل.

الطريقة	مستوى الحساب	المزايا	العيوب
التظليل المسطح	لكل مضلع	سريع جداً	غير ناعم
تظليل غورو	لكل رأس، ثم يمزج بينها	ناعم نسبياً	يفقد اللمعان
تظليل فونغ	لكل بكسل، بعد مزج المتجهات	واقعي جداً	مكلف حسابياً

الرسوم المتحركة

الرسوم المتحركة Animation : هي عملية إنشاء صور متحركة رقمية تعتمد على الحركة الوهمية للكائنات من خلال عرض الإطارات Frames تسلسلياً، هذه العملية تحول النماذج الثابتة إلى مشاهد متحركة.

بدأت الرسوم المتحركة الحاسوبية في السبعينيات مع أفلام تجريبية، ثم تطورت إلى صناعة ضخمة، حيث تم انتاج أول فيلم كرتون ثلاثي الأبعاد "Toy Story" في عام 1995.

في الوقت الحالي أصبح بإمكان الذكاء الاصطناعي توليد حركات أكثر واقعية، كما في الألعاب أو الأفلام.

أنواع الرسوم المتحركة

هناك أنواع عدة أنواع من الرسوم المتحركة، مقسمة حسب الأبعاد أو التقنيات:

1- الرسوم المتحركة ثنائية الأبعاد 2D Animation تعتمد على رسوم مسطحة، مثل أفلام الكرتون التقليدية، لكنها رقمية، فيتم رسم الإطارات Frame-by-Frame .

2- الرسوم المتحركة ثلاثية الأبعاد 3D Animation وهي الأكثر شيوعاً في الأفلام الحديثة، حيث يتم بناء نماذج ثلاثية الأبعاد وتحريكها باستخدام هياكل عظمية Rigging .

3- الرسوم المتحركة المبنية على الحركة Motion Graphics تجمع بين تصميم الجرافيك والحركة، مثل تحريك النصوص أو العناصر في الإعلانات.

4- الرسوم المتحركة المبنية على الفيزياء Physics-Based Animation تستخدم قوانين الفيزياء لمحاكاة الحركة، مثل سقوط كرة أو تدفق السوائل.

5- الرسوم المتحركة بالتقاط الحركة Motion Capture تسجل حركات ممثلين حقيقيين عبر حساسات وتطبقها على نماذج رقمية، كما في أفلام "Avatar" .

النوع	الوصف	المزايا	العيوب
2D Animation	رسوم مسطحة	سهلة ومنخفضة التكلفة	أقل واقعية
3D Animation	نماذج ثلاثية الأبعاد	واقعية عالية	تحتاج موارد حاسوبية
Motion Graphics	تحريك عناصر جرافيك	مثالية للإعلانات	محدودة في القصص المعقدة
Physics-Based	محاكاة فيزيائية	دقيقة علمياً	معقدة في الحسابات
Motion Capture	التقاط حركة حقيقية	حركات طبيعية	تحتاج معدات باهظة

خطوات إنشاء الرسوم المتحركة

1- التخطيط والقصة Storyboarding : وضع تسلسل لمشاهد القصة.

2- النمذجة : أي إنشاء نماذج ثلاثية الأبعاد.

3- الهيكلةRigging : إضافة هيكل عظمي للعناصر أو الكائنات.

4- التحريك: إنشاء Keyframes أو استخدام Motion Capture .

5- الإضاءة والتظليل: تطبيق الإضاءة لإضفاء الواقعية على المشهد.

6- Rendering : لانتاج الإطارات النهائية.

7- التعديل اللاحق Post-Production : إضافة الصوت والتأثيرات.

الأدوات والبرمجيات

Blender مجاني، يدعم الرسوم المتحركة الكاملة.

Autodesk Maya متخصص في الـ 3D، مستخدم في هوليوود.

Adobe After Effects يستخدم لتحريك الغرافيك.

Unity أو Unreal Engine للألعاب التفاعلية.

التطبيقات العملية لرسوم الحاسوب

الألعاب الأفلام، التصميم الهندسي، الطب (نماذج ثلاثية الأبعاد للأعضاء)، السينما، الواقع الافتراضي VR والواقع الافتراضي المعزز AR ، التعليم (إجراء محاكاة للدروس)، طباعة التصاميم.

التحديات و المستقبل

- الكفاءة في المصادر الحاسوبية: تتطلب عمليات الرسم المعقدة مثل الرسوم المتحركة ثلاثية الأبعاد والإضاءة الواقعية موارد حاسوبية هائلة، مما يؤدي إلى استهلاك طاقة عالي وتسخين الأجهزة، خاصة في التطبيقات المحمولة أو الأجهزة ذات القدرات المحدودة.

– تحقيق الواقعية العالية: وهذا يظل تحدياً كبيراً، إذ يتطلب محاكاة دقيقة للفيزياء الطبيعية مثل انعكاس الضوء والظلال، دون التضحية بالسرعة في الرسم الزمني الحقيقي، مما يجعل بعض التقنيات غير عملية في الألعاب أو الواقع الافتراضي.

– دمج الذكاء الاصطناعي يثير دمج الذكاء الاصطناعي مع الرسم تحديات تقنية وأخلاقية، مثل الحاجة إلى بيانات تدريب هائلة ، بالإضافة إلى المخاطر المتعلقة بحقوق الملكية الفكرية.

أما المستقبل، فيبدو مشرقاً مع ظهور تقنيات جديدة مثل الذكاء الاصطناعي والحوسبة الكمومية بحيث يمكن:

1– دمج الذكاء الاصطناعي مع تقنيات مثل تتبع الأشعة Ray Tracing ، الذي يسمح بإنتاج صور أكثر واقعية من خلال محاكاة مسار الضوء بدقة عالية، مدعوماً بالذكاء الاصطناعي لتسريع العمليات وتقليل الضوضاء في الصور.

2– إجراء تكامل أعمق مع الرؤية الحاسوبية Computer Vision .

فما المقصود بتتبع الأشعة والرؤية الحاسوبية؟

تتبع الأشعة Ray Tracing

هو تقنية متقدمة في رسوم الحاسوب تهدف إلى محاكاة سلوك الضوء الطبيعي في العالم الحقيقي بهدف إنتاج صور واقعية عالية الجودة.

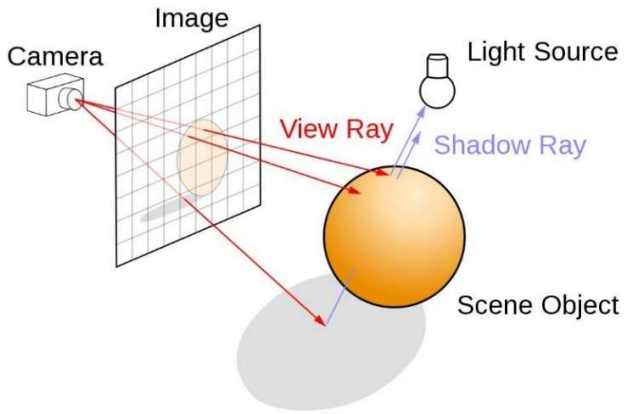
تعتمد هذه التقنية على تتبع مسار الأشعة الضوئية من الكاميرا أو المشاهد عبر المشهد، لإجراء محاكاة لتفاعلاتها مع الأسطح مثل (الانعكاس، الانتشار، والإنكسار).

بدأت فكرة تتبع الأشعة في الستينيات، لكنها مع تطور الأجهزة مثل بطاقات NVIDIA RTX في عام 2018 أصبحت عملية في الزمن الحقيقي، ومازالت هذه التقنية تتطور من خلال دمج الذكاء الاصطناعي.

كيفية عمل تتبع الأشعة

تعمل تتبع الأشعة من خلال إطلاق أشعة افتراضية من الكاميرا نحو المشهد عبر كل بكسل في الشاشة.

عند اصطدام الشعاع بكائن، يتم حساب تفاعله بناءً على خصائص السطح مثل اللون، الملمس، والانعكاس. ثم يتم إطلاق أشعة ثانوية للظلال، الانعكاسات، أو الانتشار:



1- الشعاع الأساسي من الكاميرا إلى البكسل.
2- الشعاع الظليل للتحقق من الظلال من مصادر الضوء.
3- الشعاع المنعكس، للانعكاسات.
4- الشعاع المنكسر، للشفافية.
يتكرر هذا الأمر لتحقيق الواقعية.

أنواع تتبع الأشعة

- تتبع الأشعة التقليدي Ray Casting وهو الأبسط، ويركز على الرؤية دون انعكاسات معقدة.
- تتبع الأشعة المتكرر Recursive Ray Tracing يشمل الانعكاسات المتعددة.
- تتبع المسارات Path Tracing يرسل أشعة متعددة لكل بكسل لمحاكاة الإضاءة العامة.
- تتبع الأشعة المعزز بالذكاء الاصطناعي AI-Enhanced Ray Tracing يستخدم الذكاء الاصطناعي لتحسين الأداء.

النوع	الوصف	المزايا	العيوب
التقليدي	تتبع أساسي بدون انعكاسات	سريع	غير واقعي
المتكرر	انعكاسات متكررة	واقعي أكثر	مكلف حسابياً
المسارات	محاكاة إضاءة عالمية	واقعية عالية	ضوضاء وتباطؤ
المعزز	مدعوم بالذكاء الاصطناعي	أداء محسن وصور أنقى	يعتمد على أجهزة متخصصة

الرؤية الحاسوبية Computer Vision

هي فرع من فروع الذكاء الاصطناعي يركز على تمكين الحواسيب من فهم وتفسير المحتوى البصري مثل الصور والفيديوهات، لمحاكاة الرؤية البشرية.

يمثل التكامل بين رسوم الحاسوب والرؤية الحاسوبية نقلة نوعية، تساعد على إنشاء عوالم افتراضية أكثر واقعية من خلال تحليل البيانات البصرية ودمجها مع الرسوم المتولدة.

بدأت الرؤية الحاسوبية في الخمسينيات من القرن الماضي مع تجارب أولية في التعرف على الأشكال، وتطورت بشكل كبير مع انتشار الشبكات العصبية الالتفافية CNNs في العقد الماضي، لتصبح في في الوقت الحالي تقنية أساسية في صناعات متنوعة.

في عام 2025، أصبحت الرؤية الحاسوبية مدمجة مع الذكاء الاصطناعي التوليدي Generative AI لإنتاج صور وفيديوهات واقعية، مما يعزز من تطبيقاتها في الرسوم مثل الواقع المعزز AR والافتراضي VR .

أساسيات الرؤية الحاسوبية

تعتمد الرؤية الحاسوبية على خوارزميات التعلم الآلي لمعالجة البيانات البصرية. بخطوات أساسية تشمل:

- جمع البيانات: من خلال التقاط الصور باستخدام كاميرات أو أجهزة استشعار.
- المعالجة المسبقة: تصفية الضوضاء وتعديل الإضاءة.
- استخراج الميزات: التعرف على الحواف، الألوان، والأشكال باستخدام CNNs .
- التحليل: استخدام نماذج التصنيف الآلي للكشف عن الأجسام في الزمن الحقيقي.

أنواع الرؤية الحاسوبية

تشمل الأنواع الرئيسية:

1- الكشف عن الأجسام Object Detection تحديد وتصنيف الأجسام في الصور، يستخدم مثل هذا النوع في السيارات ذاتية القيادة.
2- تصنيف الصور Image Classification أي تصنيف الصورة بأكملها، مثل هذا النوع يستخدم في التعرف على الأمراض في الصور الطبية.
3- تقسيم الصور Image Segmentation إلى أجزاء للإستفادة من ذلك في التحليل الطبي أو الزراعي.
4- تقدير الوضعية :Pose Estimation تتبع حركات الجسم، ويستخدم في الألعاب.
5- الرؤية ثلاثية الأبعاد 3D Vision إعادة بناء المشاهد بشكل ثلاثي الأبعاد (الواقع الافتراضي).

النوع	الوصف	التطبيقات الرئيسية	التحديات
الكشف عن الأجسام	كشف وتصنيف الأجسام	السيارات ذاتية القيادة	الدقة في البيئات المعقدة
تصنيف الصور	تصنيف الصورة ككل	التشخيص الطبي، الزراعة	بحاجة لبيانات تدريب هائلة
تقسيم الصور	تقسيم الصورة إلى أجزاء	التحليل الطبي، التصنيع	التكلفة الحسابية
تقدير الوضعية	تتبع الوضعيات	الألعاب، التدريب الرياضي	الحساسية للإضاءة
الرؤية ثلاثية الأبعاد	إعادة بناء مشاهد ثلاثية	الواقع المعزز	التعقيد في الدمج مع الرسوم

تقنيات الرؤية الحاسوبية

- الشبكات العصبية الالتفافية CNNs وهي الأساس لمعظم المهام.
- محولات الرؤية Vision Transformers وتستخدم لمعالجة الصور الكبيرة.
- الحوسبة الحافية Edge Computing وتستخدم لمعالجة البيانات محلياً بهدف تقليل التأخير.
- الذكاء الاصطناعي متعدد الوسائط Multimodal AI وستخدم لدمج الرؤية مع النصوص أو الصوت.

التطبيقات العملية

- الرعاية الصحية: كشف الأورام في الصور الطبية بدقة عالية.
- السيارات الذاتية: الكشف عن العوائق والإشارات.
- الزراعة: مراقبة المحاصيل وكشف الأمراض.
- التصنيع: التحكم في الجودة وكشف العيوب.
- الألعاب والترفيه: دمج الواقع المعزز مع الألعاب.

التحديات

تحتاج الرؤية الحاسوبية إلى بيانات تدريب هائلة، تكلفتها الحسابية عالية خصوصاً في تطبيقات الزمن الحقيقي، بالإضافة إلى حساسيتها للإضاءة وزوايا الرؤية، وضرورة مراعاة الخصوصية.

محاضرة (2) الخوارزميات الأساسية في الرسوم ثنائية البعد

تشكل خوازميات الرسوم ثنائية البعد 2D الأساس لفهم كيفية إنشاء ومعالجة الصور الرقمية على شاشة الحاسوب، تهتم هذه الخوارزميات برسم العناصر الأساسية مثل (الخطوط، الدوائر، والمضلعات)، مع الأخذ بعين الإعتبار كفاءة ودقة الخوارزميات خاصة في الرسوم النقطية (Raster Graphics) التي يتم تمثيلها بشبكة من البكسلات.

في هذه المحاضرة سنغطي الخوارزميات الرئيسية:

1- رسم الخطوط مثل خوارزمية (DDA و Bresenham)

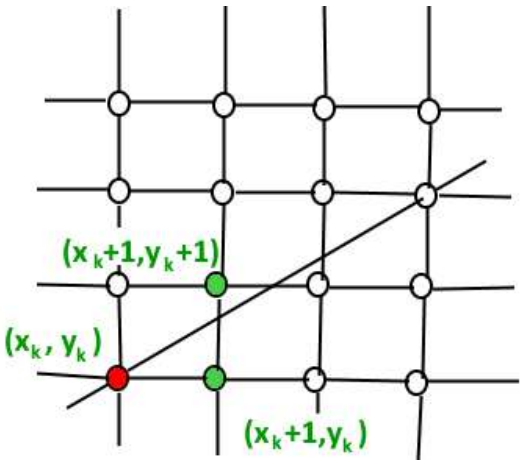
2- رسم الدوائر مثل خوارزمية (Midpoint Circle)

3- ملء المضلعات مثل خوارزمية (Scanline Fill)

تستخدم هذه الخوارزميات في الأجهزة ذات الموارد المحدودة، وكذلك الأمر في برامج مثل OpenGL ، سنشرح كل خوارزمية بالتفصيل، مع المعادلات الرياضية، الخطوات، وأمثلة عملية.

2- خوارزميات رسم الخطوط

خوارزميات رسم الخطوط المستقيمة بين نقطتين، هي من أهم الخوارزميات في الرسوم ثنائية البعد ولدينا خوارزميتان أساسيتان:



2.1. خوارزمية المحلل التفاضلي الرقمي DDA Digital Differential Analyzer

تستخدم طريقة بسيطة لرسم الخطوط، تعتمد على حساب الفرق في الإحداثيات.

بفرض أن لدينا نقطتين (x1, y1) و (x2, y2)

1- نحسب الفرق dx = x2 − x1 ، dy = y2 − y1

2- نحسب عدد الخطوات step = max(|dx|, |dy|)

3- نحسب الزيادة في كل محور: x_inc = dx / step ، y_inc = dy / step

```
def DDA(x1, y1, x2, y2):
    # نحسب الفرق في الإحداثيات
    dx = x2 - x1
    dy = y2 - y1
    # نحسب عدد الخطوات بناءً على أكبر فرق
    steps = max(abs(dx), abs(dy))
    # نحسب الزيادة في كل خطوة على المحورين
    x_inc = dx / steps
    y_inc = dy / steps
    x, y = x1, y1
    # حلقة لرسم كل بكسل
    for _ in range (steps + 1):
      # دالة لرسم البكسل
      plot(round(x), round(y))
      x += x_inc
      y += y_inc
```

هذه الخوارزمية تستخدم النقطة العائمة، مما يجعلها بطيئة في بعض الأجهزة، لكنها سلسة وسهلة الفهم.

2.2. خوارزمية بريزنهام Bresenham

تهدف هذه الخوارزمية إلى رسم خط مستقيم بين نقطتين (x0,y0) و (x1,y1) على شاشة مكونة من نقاط (Pixels) بحيث يكون الخط أقرب ما يمكن إلى الخط الرياضي الحقيقي ، بطريقة سريعة ودقيقة وبدون استخدام العمليات العشرية أو الكسور (الضرب والقسمة)، وتستخدم بدلاً من ذلك، عمليات الجمع والطرح والمقارنة، مما يجعلها مثالية للأجهزة محدودة الموارد أو الرسومات في الزمن الحقيقي.

كما نعلم فإن المعادلة الرياضية للخط المستقيم هي : $y = m.x + b$ حيث:

m هو الميل ويساوي $m = (y1-y0) / (x1-x0)$

و b هو التقاطع مع محور y

لكن هذه الصيغة تتضمن عمليات ضرب وقسمة، بدلاً من ذلك، تقوم خوارزمية بريزنهام بتقريب أقرب بكسل إلى الخط الحقيقي في كل خطوة على طول المحور الأساسي (المحور x عادةً).

<u>خطوات الخوارزمية</u>

1. نحسب القيم الابتدائية:

$dx = |x1 - x0|$, $dy = |y1 - y0|$

2. نحسب المتغير الأولي للقرار (decision parameter):

$p = 2dy - dx$

3. نبدأ من النقطة الأولى (x0, y0)

4. لكل x من x_0 إلى x_1:

نرسم النقطة الحالية (x, y)

إذا كان ($p < 0$):

تكون النقطة التالية هي (x+1, y)

نحدّث $p = p + 2dy$

أما إذا كان (p >= 0):

تكون النقطة التالية هي (x+1, y+1)

نحدّث p = p + 2(dy - dx)

الفكرة الأساسية للقرار p هو أن قيمة p تمثل مدى قرب الخط الحقيقي من البكسل الأعلى أو الأسفل، فتختار الخوارزمية البكسل الأقرب إلى المسار الرياضي الحقيقي بناءً على إشارة p.

مثال تطبيقي:

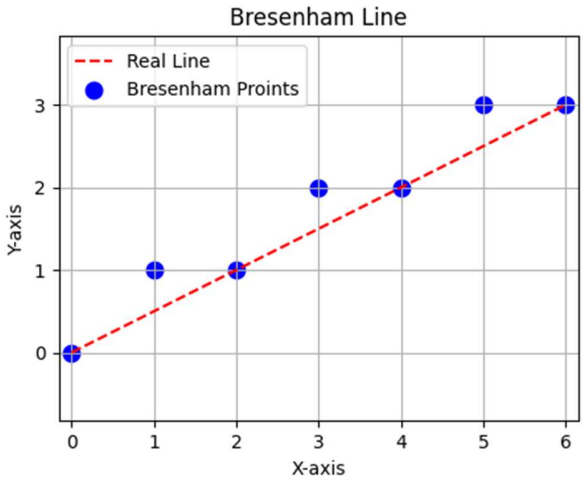
لنفترض أننا نريد رسم خط من (0، 0) إلى (6، 3):

dy = 3 ، dx = 6

p0 = 2dy - dx = 2×3 - 6 = 0

ثم نحسب باقي النقاط تدريجياً كما هو موضح بالجدول التالي:

الخطوة	x	y	p (قبل القرار)	القرار المتخذ	p (بعد التحديث)	النقطة المرسومة
0	0	0	0	$p \geq 0 \Rightarrow y+1$	$p = 0 + 2(3-6) = -6$	**(0, 0)**
1	1	1	−6	$p < 0 \Rightarrow$ لا يتغير y	$p = -6 + 2(3) = 0$	**(1, 1)**
2	2	1	0	$p \geq 0 \Rightarrow y+1$	$p = 0 + 2(3-6) = -6$	**(2, 1)**
3	3	2	−6	$p < 0 \Rightarrow$ لا يتغير y	$p = -6 + 2(3) = 0$	**(3, 2)**
4	4	2	0	$p \geq 0 \Rightarrow y+1$	$p = 0 + 2(3-6) = -6$	**(4, 2)**
5	5	3	−6	$p < 0 \Rightarrow$ لا يتغير y	$p = -6 + 2(3) = 0$	**(5, 3)**
6	6	3	—	—	—	**(6, 3)**



دالة لتنفيذ المثال السابق بلغة بايثون:

```
import matplotlib.pyplot as plt
import numpy as np

def bresenham (x0, y0, x1, y1):
    # قائمة لتخزين النقاط الناتجة
    points = []
    # حساب الفروقات بين إحداثيات النقطتين
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)
    # تحديد اتجاه الحركة على كل محور
    sx = 1 if x0 < x1 else -1
    sy = 1 if y0 < y1 else -1
    p = 2 * dy - dx  # قيمة القرار الابتدائية
    x, y = x0, y0
    while x != x1 + sx:
        points.append((x, y))
        x += sx
        if p >= 0:
            y += sy
            p += 2 * (dy - dx)
        else:
            p += 2 * dy
    return points
# ------------------ تنفيذ الدالة ------------------
x0, y0, x1, y1 = 0, 0, 6, 3
points = bresenham (x0, y0, x1, y1)
# الخط الرياضي الحقيقي
x_line = np.linspace(x0, x1, 100)
y_line = (y1 - y0)/(x1 - x0) * (x_line - x0) + y0
# رسم الخط والنقاط
plt.figure(figsize=(6,3))
plt.plot(x_line, y_line, 'r--', label='Real Line')
plt.scatter([p[0] for p in points], [p[1] for p in points],
            color='blue', s=80, label='Bresenham Proints')
plt.grid(True)
plt.axis('equal')  # توحيد مقياس المحورين
plt.title("Bresenham Line")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

3- خوارزمية رسم الدائرة Midpoint Circle Algorithm

تهدف هذه الخوارزمية إلى رسم دائرة باستخدام نقاط البكسل فقط، دون الحاجة لاستخدام دوال رياضية بهدف تقليل عدد العمليات الحسابية، تعتمد الخوارزمية على تماثل الدائرة، وبالتالي يمكن رسم 8 نقاط متماثلة من خلال حساب نقطة واحدة فقط.

معادلة الدائرة: $x^2 + y^2 = r^2$

تُستخدم دالة القرار لتحديد البكسل الذي سيتم رسمه في كل خطوة

<u>خطوات الخوارزمية</u>

1. التهيئة

- نبدأ من النقطة $(0، r)$
- القيمة الأولية لدالة القرار: $d = 1 - r$

2. التكرار، في كل خطوة،

- إذا كانت $d < 0$: نختار البكسل E (الشرق)
  - نعدل القيمة $d = d + 2x + 3$
- إذا كانت $d \geq 0$: نختار البكسل SE (الجنوب الشرقي)
  - نعدل الفيمة $d = d + 2(x - y) + 5$
  - $y = y - 1$
- $x = x + 1$

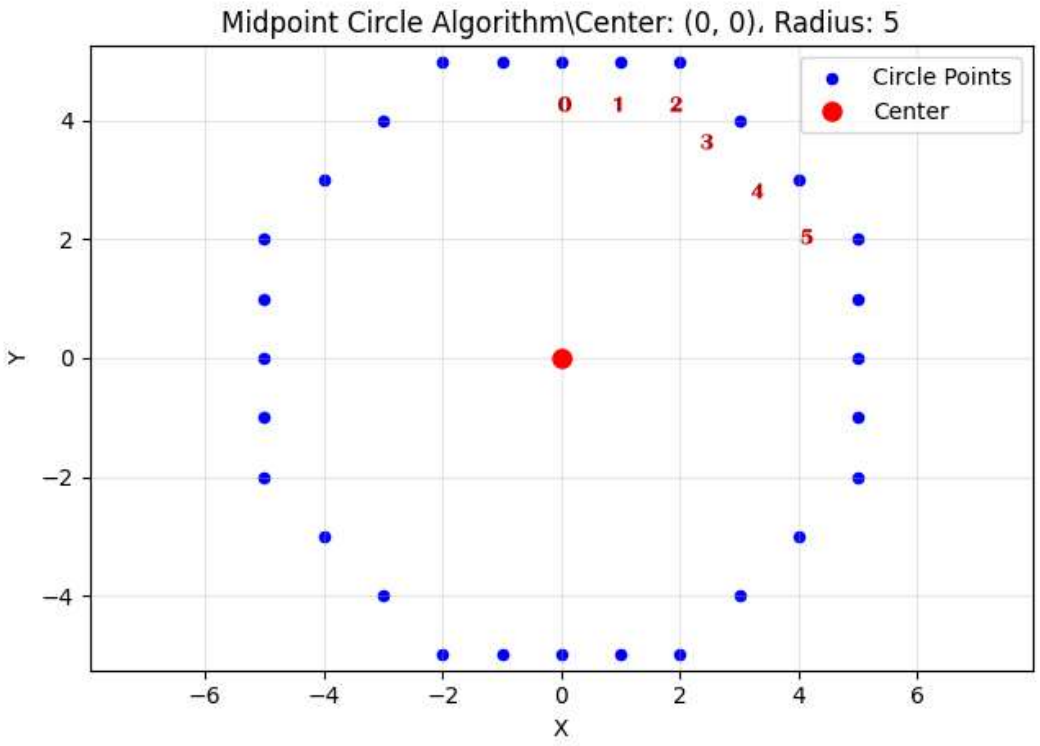
3. التماثل

- لكل نقطة (x, y) محسوبة، نرسم 8 نقاط متماثلة:
  - $(x, y), (-x, y), (x, -y), (-x, -y)$
  - $(y, x), (-y, x), (y, -x), (-y, -x)$

عندها نكون قد أكملنا الربع الأول، يكون قد تم توليد جميع النقاط الأخرى بالتماثل.

مثال تطبيقي (r = 5):

الخطوة	x	y	d	الشرط	الإجراء	القرار الجديد d	النقاط المضافة (x,y)
0	0	5	-4	-	البداية	-4	(0,5), (0,-5), (5,0), (-5,0)
1	1	5	-4	d < 0	اختيار E	-4 + 2×1 + 3 = 1	(1,5), (-1,5), (1,-5), (-1,-5), (5,1), (-5,1), (5,-1), (-5,-1)
2	2	5	1	d ≥ 0	اختيار SE	1 + 2×(2-5) + 5 = 0	(2,5), (-2,5), (2,-5), (-2,-5), (5,2), (-5,2), (5,-2), (-5,-2)
3	3	4	0	d ≥ 0	اختيار SE	0 + 2×(3-4) + 5 = 3	(3,4), (-3,4), (3,-4), (-3,-4), (4,3), (-4,3), (4,-3), (-4,-3)
4	4	3	3	d ≥ 0	اختيار SE	3 + 2×(4-3) + 5 = 10	(4,3), (-4,3), (4,-3), (-4,-3), (3,4), (-3,4), (3,-4), (-3,-4)
5	5	2	10	d ≥ 0	اختيار SE	10 + 2×(5-2) + 5 = 21	(5,2), (-5,2), (5,-2), (-5,-2), (2,5), (-2,5), (2,-5), (-2,-5)



دالة لتنفيذ المثال السابق بلغة بايثون:

```
import matplotlib.pyplot as plt

def midpoint_circle(xc, yc, r):
    points = []
    x = 0
    y = r
    d = 1 - r  # معلمة القرار الأولية
    # دالة مساعدة لإضافة النقاط الثمانية المتماثلة
    def plot_symmetric_points(x, y):
        points.extend([
            (xc + x, yc + y), (xc - x, yc + y),
            (xc + x, yc - y), (xc - x, yc - y),
            (xc + y, yc + x), (xc - y, yc + x),
            (xc + y, yc - x), (xc - y, yc - x)
        ])
    # إضافة النقاط الأولية
    plot_symmetric_points(x, y)
    # معالجة الربع الأول فقط (من 0° إلى 45°)
    while x < y:
        if d < 0:
            # تحريك أفقيًا (شرقًا)
            d += 2 * x + 3
            x += 1
        else:
            # تحريك قطريًا (جنوب شرق)
            d += 2 * (x - y) + 5
            x += 1
            y -= 1
        plot_symmetric_points(x, y)
    # إزالة التكرارات وترتيب النقاط
    unique_points = sorted(set(points))
    return unique_points
# ============== تشغيل المثال ==============
# إعدادات الدائرة
center_x, center_y = 0, 0
radius = 15
# توليد نقاط الدائرة
circle_points = midpoint_circle(center_x, center_y, radius)
# فصل الإحداثيات
x_coords, y_coords = zip(*circle_points)
# matplotlib رسم الدائرة باستخدام
plt.figure(figsize=(8, 8))
```

```
plt.scatter(x_coords, y_coords, color='blue', s=20, label='Circle Points')
plt.plot(center_x, center_y, 'ro', markersize=8, label='Center')
plt.title(f'Midpoint Circle Algorithm\Center: ({center_x}, {center_y}) ،Radius:
{radius}')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, alpha=0.3)
plt.axis('equal')
plt.legend()
plt.show()
```

4- خوارزميات ملء المضلعات Scanline Fill Algorithm

خوارزمية Scanline Fill (الملء بمسح الخطوط) تستخدم هذه الخوارزمية لملء المضلعات في رسومات الحاسوب، تعتمد على فكرة مسح الشكل خطاً تلو الآخر، وتحديد نقاط التقاطع مع حواف المضلع.

تعتمد الخوارزمية على: مسح الخطوط الأفقية من الأعلى إلى الأسفل، تحديد نقاط التقاطع مع حواف المضلع، تعبئة الأجزاء الداخلية :بين نقاط التقاطع.

خطوات الخوارزمية

1. إيجاد حواف المضلع
   - تخزين جميع حواف المضلع في جدول الحواف (Edge Table – ET) .
2. إنشاء جدول الحواف النشطة (Active Edge Table – AET)
   - يحتوي على الحواف التي تتقاطع مع خط المسح الحالي
3. عملية المسح
   - من أعلى نقطة (أصغر y) إلى أسفل نقطة (أكبر y).
   - لكل خط مسح:
     - إضافة الحواف التي تبدأ عند هذا الخط إلىAET
     - إزالة الحواف التي تنتهي عند هذا الخط منAET
     - ترتيب الحواف في AET حسب إحداثيات x
     - تعبئة البكسلات بين أزواج نقاط التقاطع

مثال تطبيقي:

مطلوب تعبئة المضلع المحدد بالإحداثيات التالية:

A(2,1), B(6,1), C(8,5), D(6,8), E(2,8), F(4,5)

الخطوة 1: بناء جدول الحواف ET ، نرتّب الحواف تصاعدياً حسب قيمة ymin

Edge	from → to	ymin	ymax	x @ ymin	inv_slope (dx/dy)
BC	(6,1)→(8,5)	1	5	6	+0.5
FA	(2,1)→(4,5)	1	5	2	+0.5
CD	(8,5)→(6,8)	5	8	8	−0.6666667
EF	(4,5)→(2,8)	5	8	4	−0.6666667

الخطوة 2: حساب نقاط التقاطع مع كل عملية مسح

تحسب نقاط التقاطع وفق العلاقة التالية: $\text{Cross x} = \mathbf{x\ @\ ymin} + \mathbf{inv\_slope} \times (\text{y\_scan} - \mathbf{ymin})$

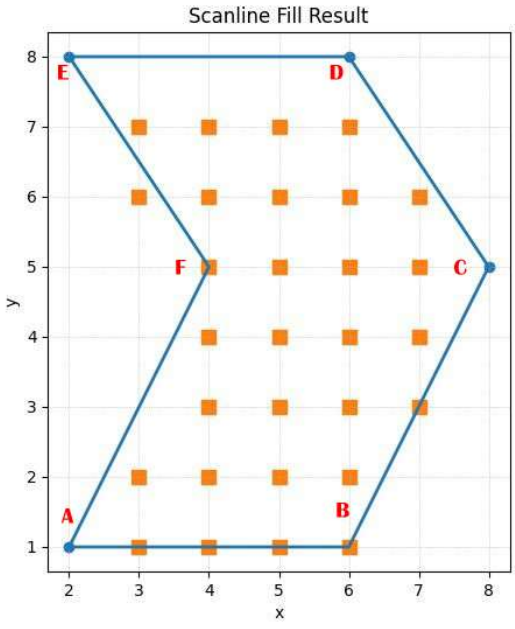
عند خط المسح y_scan = 1.5 تكون نقاط التقاطع:

FA x = 2 + 0.5 ×(1.5 - 1) = 2.25

BC x = 6 + 0.5 ×(1.5 - 1) = 6.25

وبالتالي نقاط التعبئة هي : 3، 4، 5، 6

نحسب باقي النقاط بنفس الطريقة، لنحصل على جدول الحواف النشطة AET:



y	y_scan	تقاطع X عند	بكسلات مملوءة y على
1	1.5	2.25 , 6.25	3,4,5,6
2	2.5	2.75 , 6.75	3,4,5,6
3	3.5	3.25 , 7.25	4,5,6,7
4	4.5	3.75 , 7.75	4,5,6,7
5	5.5	3.67 , 7.67	4,5,6,7
6	6.5	3.0 , 7.0	3,4,5,6,7
7	7.5	2.33 , 6.33	3,4,5,6

دالة لتنفيذ المثال السابق بلغة بايثون:

```
import math
import matplotlib.pyplot as plt

def scanline_fill(polygon):

    #حساب أعلى وأسفل y للمضلّع
    ys = [p[1] for p in polygon]
    y_min = math.floor(min(ys))
    y_max = math.ceil(max(ys))

    filled_pixels = []

    #نعمل مسحًا لكل صف y (نأخذ نقطة المسح عند y + 0.5)
    for y in range(y_min, y_max):
        y_scan = y + 0.5

        # جمع نقاط تقاطع الخط مع حواف المضلّع
        x_intersections = []
        n = len(polygon)
        for i in range(n):
            x1, y1 = polygon[i]
            x2, y2 = polygon[(i + 1) % n]

            # تجاهل الحواف الأفقية
            if y1 == y2:
                continue

            # نتحقق إن كانت نقطة المسح تقع داخل مدى y
            y_min_edge = min(y1, y2)
            y_max_edge = max(y1, y2)
            if (y_scan >= y_min_edge) and (y_scan < y_max_edge):
                # نحسب إحداثيات x للتقاطع
                t = (y_scan - y1) / (y2 - y1)
                x_int = x1 + t * (x2 - x1)
                x_intersections.append(x_int)

        # إذا لم يوجد تقاطعات، نتخطّى هذا الصف
        if not x_intersections:
            continue

        # نرتّب نقاط التقاطع ونأخذها زوجًا زوجًا
        x_intersections.sort()
```

```
        # يجب أن تكون التقطعات زوجية
        for i in range(0, len(x_intersections), 2):
            x_left = x_intersections[i]
            # قد يحدث، لأسباب رقمية، أن لا يوجد زوج كامل؛ نتفادى ذلك
            if i+1 >= len(x_intersections):
                break
            x_right = x_intersections[i+1]
            # نحول النطاق العائم إلى بكسلات صحيحة
            x_start = math.ceil(x_left)
            x_end = math.floor(x_right)
            # y نضيف كل بكسيل داخل النطاق [x_start, x_end] عند الصف
            for x in range(x_start, x_end + 1):
                filled_pixels.append((x, y))
    return filled_pixels
```

محاضرة (3) التحويلات الأساسية في الرسوم ثنائية الأبعاد

تُعد التحويلات الأساسية الرسوم ثنائية البعد من العمليات الجوهرية التي تتيح تعديل الكائنات الرسومية من حيث الموقع، الحجم، والتوجه.

تشمل هذه التحويلات الإزاحة رسوم Translation، التكبير/التصغير Scaling، والدوران Rotation، وأحياناً القص Shearing.

تُستخدم هذه العمليات لتحريك الأشكال، تغيير أحجامها، أو تدويرها في الفضاء ثنائي الأبعاد، وهي أساسية في تطبيقات مثل الألعاب، برامج التصميم، والرسوم المتحركة.

تعتمد هذه التحويلات على الرياضيات الخطية، وخاصة مصفوفات التحويل، التي تتيح تسلسل العمليات بكفاءة.

في هذا القسم، سنتوسع في شرح هذه التحويلات، مع التركيز على الصيغ الرياضية، تنفيذها برمجياً، والتحديات المرتبطة بها، مع بعض الأمثلة العملية لتوضيح كيفية تطبيق هذه التحويلات في سياقات مختلفة.

التطبيقات العملية

- ✓ **الألعاب** : تحريك الشخصيات أو الكاميرا.
- ✓ **برامج التصميم** : مثل Adobe Illustrator لتعديل الأشكال.
- ✓ **الرسوم المتحركة** : لإنشاء حركات سلسة.
- ✓ **CAD** : لتصميم الأجزاء بدقة.

التحديات

- ☹ **التراكم الرقمي** : العمليات العائمة قد تؤدي إلى أخطاء تراكمية عند التكرار.
- ☹ **الأداء** : الدوران والقص يتطلبان عمليات مثلثية مكلفة. حسابياً
- ☹ **التحويل حول نقطة غير الأصل** : يتطلب إزاحة الأصل، تطبيق التحويل، ثم إعادة الإزاحة.

أنواع التحويلات الأساسية

1- الإزاحة Translation

الإزاحة هي تحريك كائن من موقع إلى آخر دون تغيير حجمه أو توجهه، يتم تمثيل النقطة x,y بعد الإزاحة بـ x', y' كالتالي:

$$x' = x + tx$$

$$y' = y + ty$$

حيث tx و ty هما مقدار الإزاحة في المحورين x وy

الخوارزمية:

```
def Translate(x, y, tx, ty):
    # إضافة مقدار الإزاحة مباشرة إلى الإحداثيات
    x_new = x + tx
    y_new = y + ty
    return x_new, y_new
```

هذه العملية بسيطة للغاية، حيث تضيف قيم الإزاحة إلى الإحداثيات الأصلية.

مثال ذلك: إذا كانت النقطة (2، 3) و tx=5، ty=4 فإن النقطة الجديدة هي (8, 6).

2- التكبير/التصغير Scaling

التكبير يغير حجم الكائن دون تغيير موقعه النسبي أو توجهه، يتم ضرب الإحداثيات في معاملات التكبير.

الصيغة:

$$x' = x * sx$$

$$y' = y * sy$$

حيث sx و sy هما معاملات التكبير في المحورين x و y

مثال ذلك: إذا كانت النقطة (3، 2) و sx=2 ، sy=0.5 فإن النقطة الجديدة هي (4, 1.5)

الخوارزمية:

```
def Scale(x, y, sx, sy):
    # ضرب الإحداثيات في معاملات التكبير
    x_new = x * sx
    y_new = y * sy
    return x_new, y_new
```

إذا كان $sx > 1$ ، يزداد الحجم، وإذا كان $sx < 1$ ، ينخفض.

إذا كان $sx \neq sy$ ، يحدث تكبير غير متجانس، مما قد يشوه الشكل (مثلاً تحويل دائرة إلى شكل بيضوي).

3- الدوران Rotation

الدوران يغير توجه الكائن حول نقطة مرجعية، عادة الأصل (0,0) ، يتم استخدام الزاوية θ لتحديد مقدار الدوران.

الصيغة:

$$x' = x * \cos(\theta) - y * \sin(\theta)$$

$$y' = x * \sin(\theta) + y * \cos(\theta)$$

حيث θ هي زاوية الدوران.

الخوارزمية:

```
def Rotate(x, y, theta):
# استخدام الدوال المثلثية لتدوير النقطة
    x_new = x * cos(theta) - y * sin(theta)
    y_new = x * sin(theta) + y * cos(theta)
    return x_new, y_new
```

العمليات المثلثية sin و cos مكلفة حسابياً، لذا يُفضل تخزين القيم مسبقاً للزوايا الشائعة.

مثال ذلك : إذا كانت النقطة (0، 1) و زاوية الدوران θ=90°، فإن النقطة الجديدة هي تقريباً (0,1).

4- القص Shearing

القص يشوه الكائن باتجاه معين، مما يغير شكله دون تغيير حجمه.

الصيغة:

$$x' = x + shx * y$$

$$y' = y + shy * x$$

حيث shx و shy هما معاملا القص.

الخوارزمية:

```
def Shear(x, y, shx, shy):
    # إضافة تأثير القص بناءً على الإحداثيات الأخرى
    x_new = x + shx * y
    y_new = y + shy * x
    return x_new, y_new
```

مثال: إذا كانت النقطة (2 ، 1) و shx=0.5 ، shy=0 فإن النقطة الجديدة هي (2 , 2)

قد يؤدي القص إلى تشوه غير مرغوب إذا لم يتم التحكم في معاملاته بدقة.

5- الدوران حول نقطة معينة

الخوارزمية:

```
def RotateAroundPoint(x, y, cx, cy, theta):
    # (cx, cy) نقل النقطة إلى الأصل نسبة إلى
    x_temp = x - cx
    y_temp = y - cy
    # تطبيق الدوران
    x_new = x_temp * cos(theta) - y_temp * sin(theta) + cx
    y_new = x_temp * sin(theta) + y_temp * cos(theta) + cy
    return x_new, y_new
```

يجب التأكد من دقة النقطة المرجعية (cx, cy) لتجنب انحراف الدوران.

ما تم شرحة من تحويلات هو لنقطة واحدة فقط ، الآن سنرى كيف يمكن تعميم التحويلات لأكثر من نقطة.

6- التعميم

6-1- تعميم الإزاحة لـ مستقيم:

استخدمانا الدالة Translate لإزاحة نقطة ، المستقيم مكون من نقطتين (x1,y1) ، (x2,y2) نُزيح كل نقطة باستخدام نفس الدالة.

```
def TranslateLine(x1, y1, x2, y2, tx, ty):
# إزاحة النقطة الأولى باستخدام نفس الدالة
    x1_new, y1_new = Translate(x1, y1, tx, ty)
# إزاحة النقطة الثانية باستخدام نفس الدالة
    x2_new, y2_new = Translate(x2, y2, tx, ty)
# النتيجة
    return x1_new, y1_new, x2_new, y2_new
```

6-2- تعميم الإزاحة لـ مضلع:

المضلع = قائمة من النقاط points ، نُزيح كل نقطة في القائمة باستخدام Translate

points: [(x1,y1), (x2,y2), ...], tx, ty: مقدار الإزاحة

```
def TranslatePolygon(points, tx, ty):
  # قائمة فارغة لتخزين النقاط الجديدة
    translated_points = []
  # حلقة على كل نقطة في المضلع
    for x, y in points:
      # إزاحة النقطة الحالية باستخدام نفس الدالة
        x_new, y_new = Translate(x, y, tx, ty)
      # إضافة النقطة الجديدة إلى القائمة
        translated_points.append((x_new, y_new))
    return translated_points
```

6-3- تعميم التكبير / التصغير لـ مضلع Polygon

المضلع = قائمة من النقاط points: [(x1,y1), (x2,y2), ...]

نُكبّر / نصغر كل نقطة في القائمة بالمعاملات sx, sy بالنسبة للأصل (0,0)

ترجع: قائمة جديدة من النقاط المُكبّرة أو المصغرة

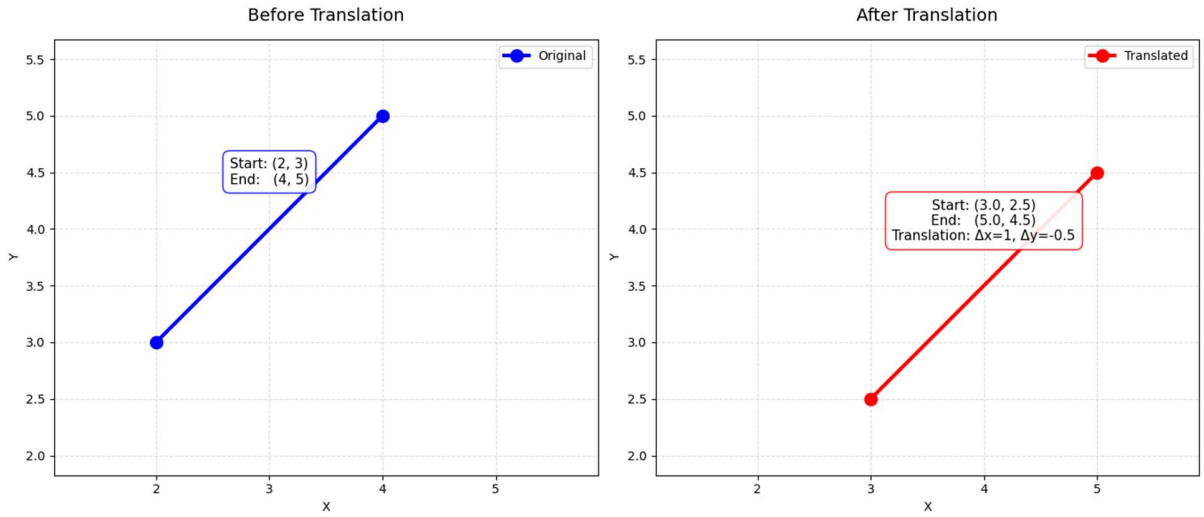
```
def ScalePolygon(points, sx, sy):
   scaled_points[] =
```

```
    for x, y in points:
        x_new, y_new = Scale(x, y, sx, sy)
        scaled_points.append((x_new, y_new))
    return scaled_points
```

دالة لإزاحة مستقيم:

```
# ------------------------------------------------
# 1. Translation functions
# ------------------------------------------------
def Translate(x, y, tx, ty):
    """Translate point (x, y) by (tx, ty)."""
    return x + tx, y + ty

def TranslateLine(x1, y1, x2, y2, tx, ty):
    """Translate line segment."""
    x1n, y1n = Translate(x1, y1, tx, ty)
    x2n, y2n = Translate(x2, y2, tx, ty)
    return x1n, y1n, x2n, y2n

# ------------------------------------------------
# 2. Short line + small translation
# ------------------------------------------------
x1, y1 = 2, 3      # start point (shorter line)
x2, y2 = 4, 5      # end point   (length = 2√2 ≈ 2.8)
tx, ty = 1, -0.5   # small translation

# Translated line
x1n, y1n, x2n, y2n = TranslateLine(x1, y1, x2, y2, tx, ty)

# ------------------------------------------------
# 3. All points
# ------------------------------------------------
all_x = np.array([x1, x2, x1n, x2n])
all_y = np.array([y1, y2, y1n, y2n])

# ------------------------------------------------
# 4. Axis limits with 40% padding
# ------------------------------------------------
margin = 0.4  # 40% padding
x_min, x_max = all_x.min(), all_x.max()
y_min, y_max = all_y.min(), all_y.max()
```

```
x_range = max(x_max - x_min, 1)
y_range = max(y_max - y_min, 1)

x_lim = [x_min - x_range * margin, x_max + x_range * margin]
y_lim = [y_min - y_range * margin, y_max + y_range * margin]
```

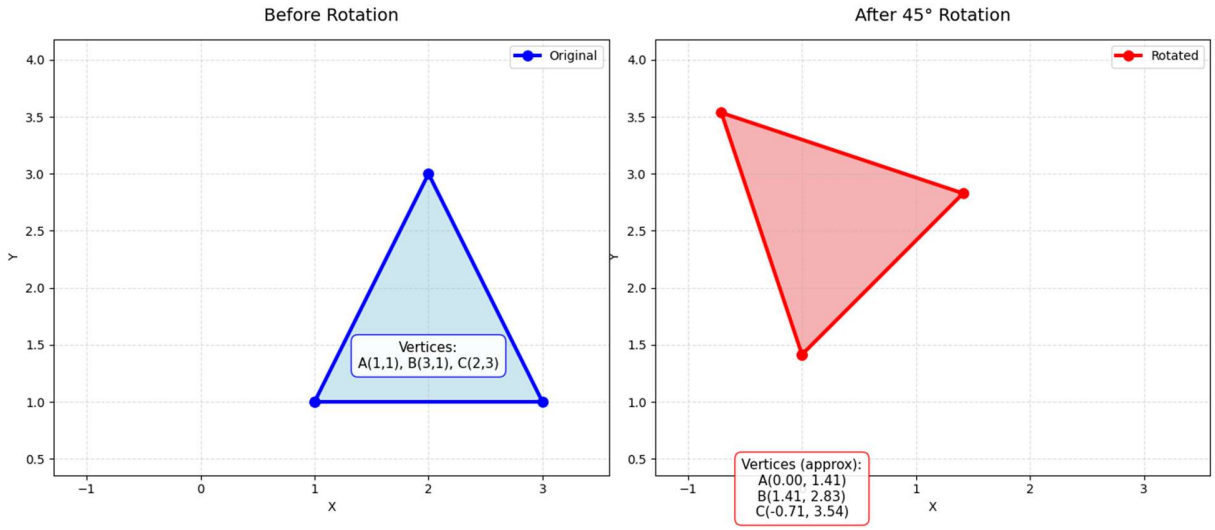


دالة لدوران مثلث:

```
# ------------------------------------------------
# 1. Rotation of a single point (around origin)
# ------------------------------------------------
def Rotate(x, y, theta_deg):
    """
    Rotate point (x, y) by theta_deg degrees (counter-clockwise) around (0,0).
    Returns (x_new, y_new).
    """
    theta = radians(theta_deg)
    x_new = x * cos(theta) - y * sin(theta)
    y_new = x * sin(theta) + y * cos(theta)
    return x_new, y_new

# ------------------------------------------------
# 2. Rotate a triangle (list of 3 points)
# ------------------------------------------------
def RotateTriangle(points, theta_deg):
    """
```

```
    Rotate a triangle given as list of tuples: [(x1,y1), (x2,y2), (x3,y3)]
    Returns new list of rotated points.
    """
    rotated = []
    for x, y in points:
        x_new, y_new = Rotate(x, y, theta_deg)
        rotated.append((x_new, y_new))
    return rotated

# ------------------------------------------------
# 3. Original triangle (small, centered near origin)
# ------------------------------------------------
triangle = [
    (1.0, 1.0),   # vertex A
    (3.0, 1.0),   # vertex B
    (2.0, 3.0)    # vertex C
]

theta_deg = 45  # rotation angle

# ------------------------------------------------
# 4. Rotated triangle
# ------------------------------------------------
rotated_triangle = RotateTriangle(triangle, theta_deg)

# ------------------------------------------------
# 5. Prepare data for plotting (close the polygon)
# ------------------------------------------------
def close_polygon(pts):
    x, y = zip(*pts)
    return list(x) + [x[0]], list(y) + [y[0]]

orig_x, orig_y = close_polygon(triangle)
rot_x,  rot_y  = close_polygon(rotated_triangle)

# ------------------------------------------------
# 6. Same axis limits for both sub-plots
# ------------------------------------------------
all_x = np.array(orig_x + rot_x)
all_y = np.array(orig_y + rot_y)

margin = 0.4
x_min, x_max = all_x.min(), all_x.max()
y_min, y_max = all_y.min(), all_y.max()
```

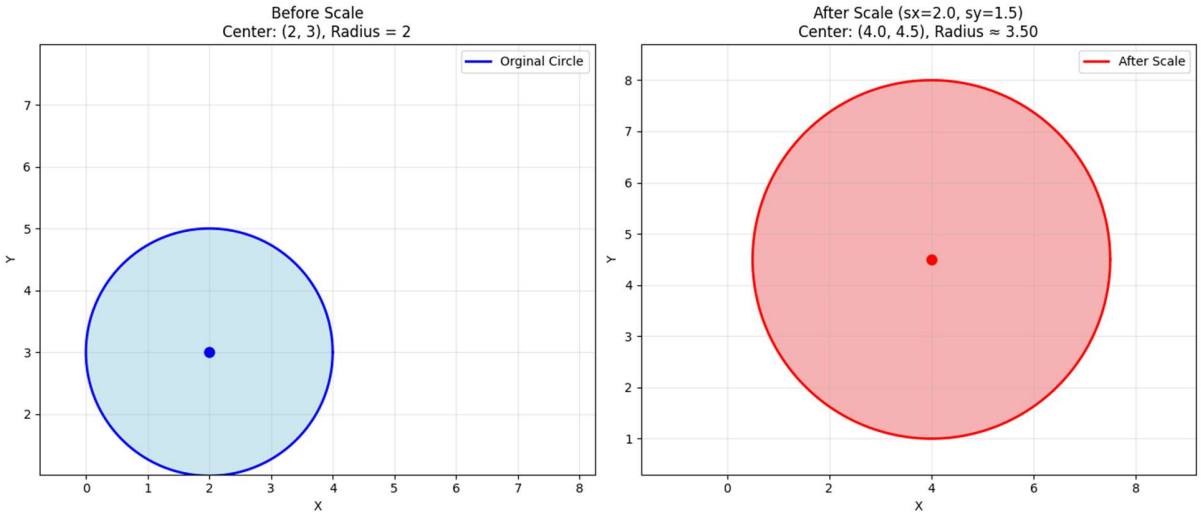
```
x_range = max(x_max - x_min, 1)
y_range = max(y_max - y_min, 1)

x_lim = [x_min - x_range * margin, x_max + x_range * margin]
y_lim = [y_min - y_range * margin, y_max + y_range * margin]
```



دالة لتغير حجم دائرة:

```
import matplotlib.pyplot as plt
import numpy as np
# ------------------------------
# 1. الدوال الأساسية
# ------------------------------
def Scale(x, y, sx, sy):
    """تكبير نقطة"""
    return x * sx, y * sy

def ScaleCircle(cx, cy, r, sx, sy):
    """تكبير دائرة (تقريب دائري)"""
    cx_new, cy_new = Scale(cx, cy, sx, sy)
    r_new = r * ((sx + sy) / 2)       # متوسط التكبير
    return cx_new, cy_new, r_new
# ------------------------------
# 2. دائرة أصلية
# ------------------------------
cx, cy = 2, 3
```

```
r = 2
# 1.5× رأسي ،2× تكبير أفقي
sx, sy = 2.0, 1.5

# --------------------------------
# 3. تكبير الدائرة
# --------------------------------
cx_new, cy_new, r_new = ScaleCircle(cx, cy, r, sx, sy)

# --------------------------------
# 4. إنشاء نقاط المحيط
# --------------------------------
theta = np.linspace(0, 2*np.pi, 200)

# الأصلية
x_orig = cx + r * np.cos(theta)
y_orig = cy + r * np.sin(theta)

# بعد التكبير
x_new = cx_new + r_new * np.cos(theta)
y_new = cy_new + r_new * np.sin(theta)

# --------------------------------
# 5. حساب حدود المحاور الموحدة
# --------------------------------
all_x = np.concatenate([x_orig, x_new])
all_y = np.concatenate([y_orig, y_new])

x_min, x_max = all_x.min(), all_x.max()
y_min, y_max = all_y.min(), all_y.max()

# %10 إضافة هامش
margin_x = (x_max - x_min) * 0.1
margin_y = (y_max - y_min) * 0.1

x_lim = [x_min - margin_x, x_max + margin_x]
y_lim = [y_min - margin_y, y_max + margin_y]
```


Before Scale
Center: (2, 3), Radius = 2
Orginal Circle
X
Y
After Scale (sx=2.0, sy=1.5)
Center: (4.0, 4.5), Radius ≈ 3.50
After Scale
X
Y

محاضرة (4) النمذجة

الهدف من هذه المحاضرة هو فهم ماهية النمذجة، التعرف على أنواع النمذجة ثلاثية الأبعاد، كيفية استخدامها في الألعاب، الأفلام، الهندسة، الطب، وما هي الأدوات والتقنيات الحديثة المستخدمة في هذا القطاع.

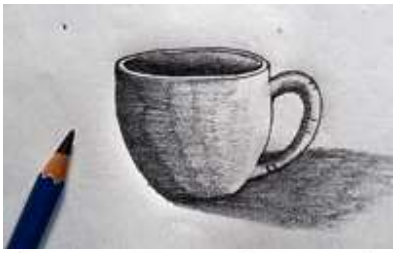
النمذجة: هي عملية إنشاء تمثيل رقمي لكائن حقيقي أو خيالي باستخدام الحاسوب.

النوع	الوصف
نقطة	(x, y, z)
مستقيم	نقطتان
مضلع	3 نقاط أو أكثر
سطح	مجموعة مضلعات
كائن ثلاثي الأبعاد	مجموعة أسطح

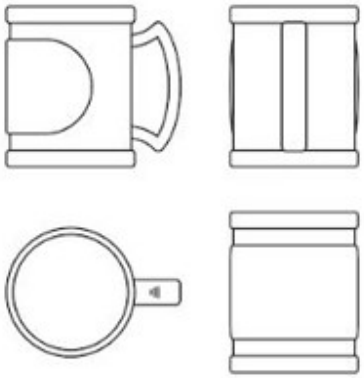
الخطوات الأساسية للنمذجة:

الفكرة ◄ رسم تخطيطي ◄ النمذجة الأولية ◄ النمذجة التفصيلية ◄ التظليل ◄ الهيكلة ◄ الإخراج

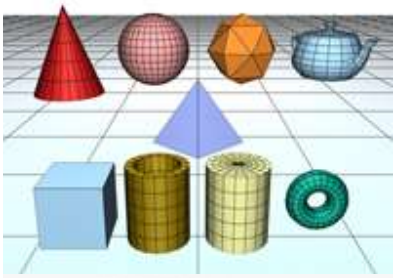
1 الفكرة: ما هو الغرض الذي نريد نمذجته (كوب على سبيل المثال) ممكن رسم الفكرة بالورقة والقلم.



2 رسم تخطيطي (Sketch / Blueprint): رسومات توضيحية للشكل من زوايا مختلفة.



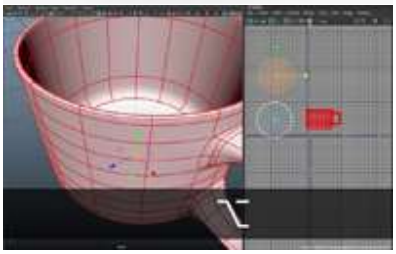
3 النمذجة الأولية (Blocking / Base Mesh): بناء الشكل العام باستخدام أشكال بدائية Primitives (مكعب، أسطوانة) لتحديد الحجم والتناسب الأساسي للنموذج.



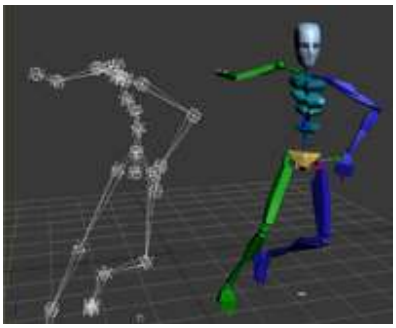
4 النمذجة التفصيلية (Detailing / High-Poly): تحسين الشكل وإضافة التفاصيل الدقيقة مثل مقبض الكوب، سمك الجدران، حواف فم الكوب، والتأكد من طوبولوجيا النموذج (أي سلامة توزيع المضلعات).

5 التظليل UV Mapping: تحويل المجسم إلى خريطة من المسطحات ثنائية البعد Texture باستخدام UV والتي هي عبارة عن إحداثيات ثنائية الأبعاد (مثل x,y) تُطبق على الأسطح ثلاثية الأبعاد.

إضافة الألوان (اللمعة، الخشونة) لخرائط النسيج باستخدام الـ Materials لإعطاؤه مظهرًا واقعيًا.



6 الهيكلة Rigging: إضافة هيكل عظمي (Skeleton) لتحريك النماذج المتحركة.

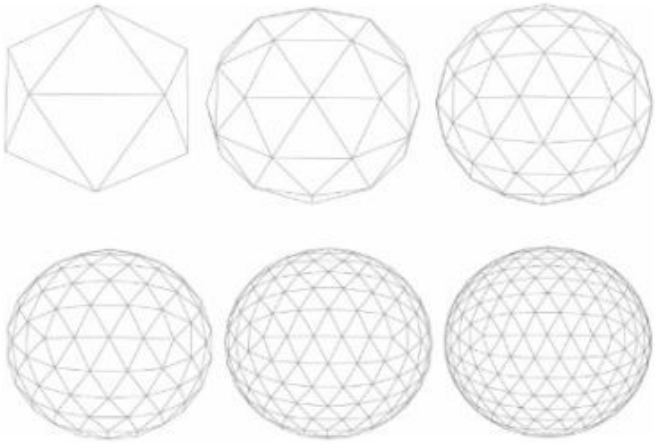


7 الإخراج (Rendering): هي عملية تحويل المشهد ثلاثي الأبعاد بالكامل (النماذج، الإضاءة، المواد) إلى صورة أو فيلم ثنائي الأبعاد.

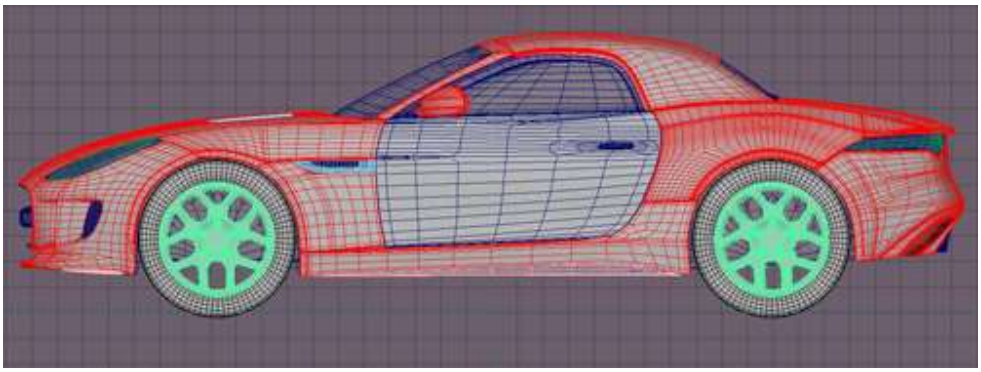
2- أنواع النمذجة:

هناك عدة أساليب، كل منها يناسب مشاريع وتقنيات مختلفة...

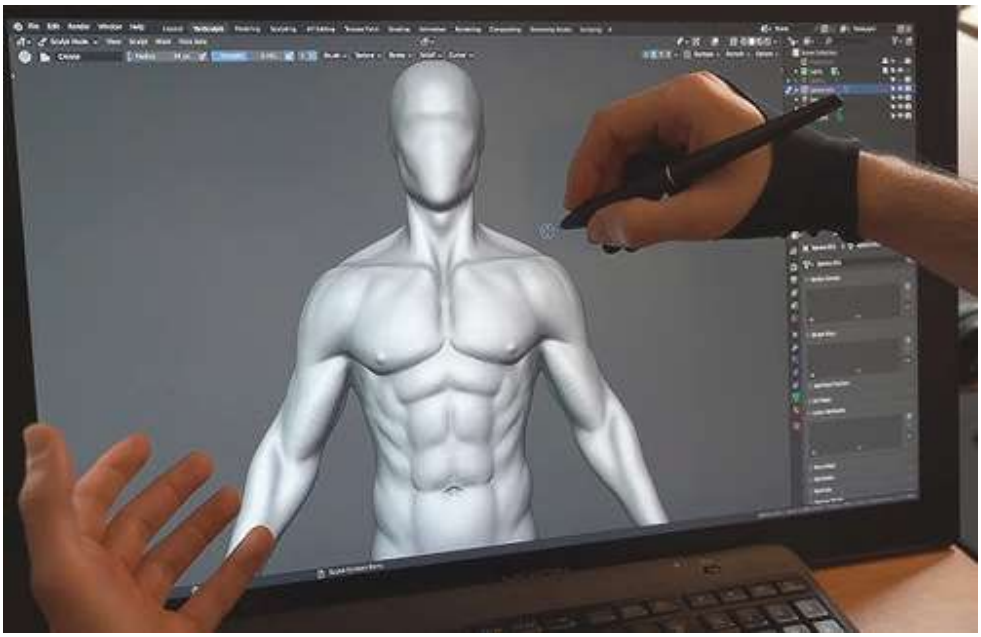
أ. نمذجة المضلعات (Polygonal Modeling) وهي الأكثر شيوعاً واستخداماً، خاصة في ألعاب الفيديو والرسوم المتحركة، ويعتمد بناؤها على الرؤوس والحواف والأوجه، وهي مرنة ومدعومة بشكل واسع إلا أنها تحتاج إلى عدد كبير من المضلعات High Poly لإنشاء أسطح ملساء.



ب. نمذجة الأسطح الرياضية الناعمة NURBS (Non-Uniform Rational B-Splines) وهذا النوع يعتمد على المنحنيات الرياضية (Curves) لوصف الأسطح، مما يسمح ببناء أسطح ناعمة ودقيقة جداً، دون الحاجة لكثافة عالية من المضلعات، وتستخدم في تصميم السيارات، المنتجات الصناعية، والأسطح العضوية المعقدة.



ج. نمذجة النحت الرقمي (Digital Sculpting) وهي تشبه النحت الحقيقي في الطين الرقمي، تسمح بإنشاء تفاصيل عضوية عالية الدقة (مثل التجاعيد، العضلات، الطيات).



د. النمذجة الإجرائية (Procedural Modeling) وتعتمد على القواعد والخوارزميات لتوليد النماذج تلقائياً، وتستخدم لإنشاء: المناظر الطبيعية، المباني، الأنماط المعقدة، والمحتوى العشوائي.

النوع	الوصف	أدوات النمذجة	الاستخدام
Polygon	نمذجة المضلعات (مثلثات)	Maya	**الألعاب، الأفلام**

NURBS	نمذجة الأسطح الرياضية الناعمة	Rhino	**السيارات، الطائرات**
Sculpting	نمذجة النحت الرقمي	ZBrush	**الشخصيات، التماثيل**
Procedural	نمذجة الخوارزميات	Houdini	**المدن، الغابات**

1- نمذجة المضلعات

هي عملية بناء الأجسام ثلاثية الأبعاد باستخدام المضلعات Polygons والمضلعات هي أشكال هندسية مسطحة تتكون عادة من رؤوس Vertices و حواف Edges و أوجه Faces.

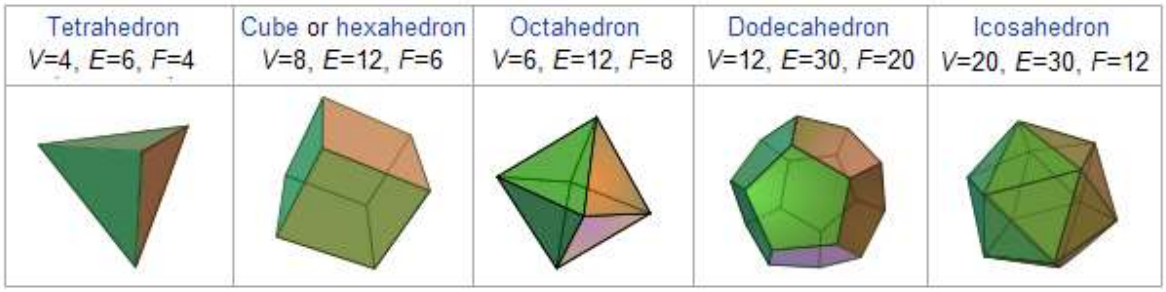
الرؤوس Vertices : الرأس هو نقطة فردية في فضاء ثلاثي الأبعاد (إحداثياته X, Y, Z) .

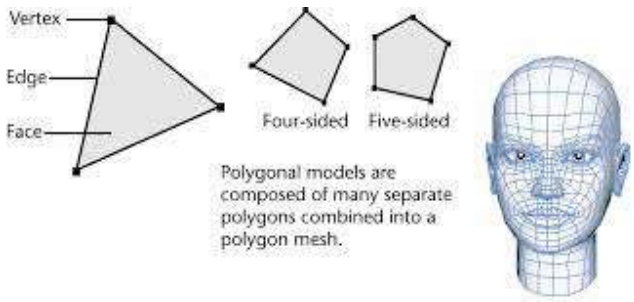
الحواف Edges : خط مستقيم يربط بين رأسين، وهي تحدد حدود وشكل النموذج.

الوجوه Faces : وهي المسطح المغلق الذي يتشكل عند ربط ثلاثة حواف أو أكثر، وهذه "الأسطح" هي التي نراها فعلياً في النموذج النهائي.

المضلع Polygon : هو مجموعة من الأوجه، متصلة ببعضها وتشكل جزءً من الشكل النهائي.

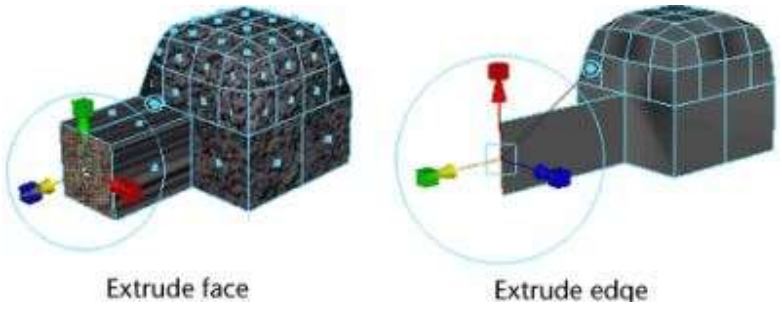
الشبكة Mesh : هي الجسم الكامل المكوّن من المضلعات المتصلة ببعضها.



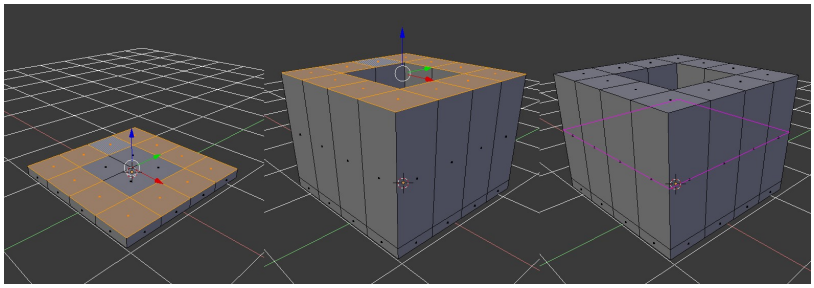


أهم الأدوات التي نستخدمها في تشكيل الأغراض في نمذجة المضلعات :

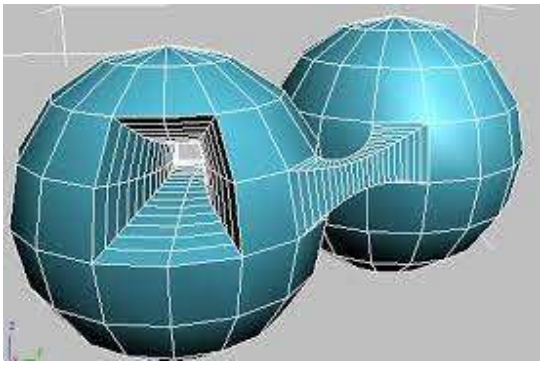
1- الإنبثاق Extrude : ويعني أخذ وجه أو مجموعة أوجه وسحبها للخارج مع إنشاء وجوه جديدة على الجوانب، مثلاً لصنع ذراع من صندوق نقوم ببثق الوجه الجانبي.



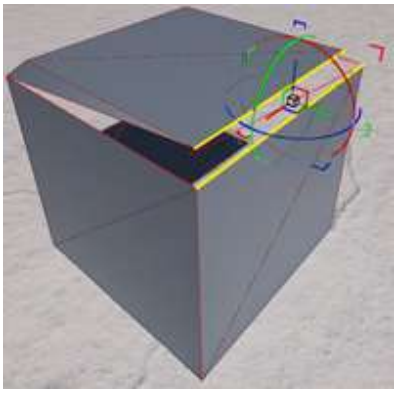
2- **الحلقة Loop Cut & Slide** : إضافة حلقة من الحواف الجديدة حول النموذج



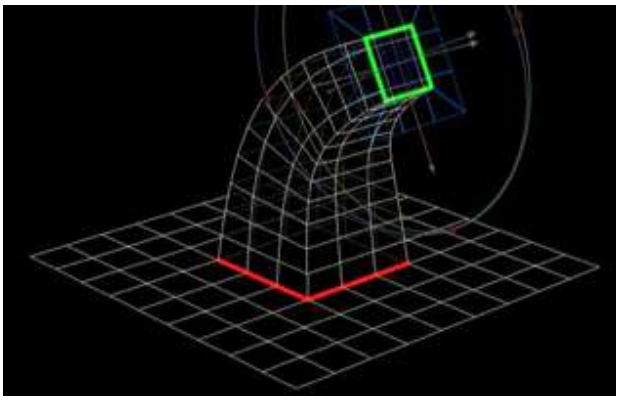
3- الربط Bridge : ربط حافتين أو وجهين ببعضهما لإنشاء نفق أو جسر بينهما.



4- اللحام Weld/Vertices Merge : دمج رأسين أو أكثر في رأس واحد لتقليل التعقيد أو إغلاق الفجوات.



5- **العزل Extrude & Scale** : طريقة تستخدم لإنشاء حواف حادة عن طريق البثق ثم تحجيم الوجه الجديد إلى الداخل.



طوبولوجيا النمذجة المضلعة هي ليست مجرد مضلعات وإنما هي طريقة لترتيب وتوزيع الرؤوس والحواف والوجوه على سطح النموذج ثلاثي الأبعاد (طوبولوجيا: خصائص الفراغات الثابتة تحت أي تشوه مستمر).

أهمية طوبولوجيا النمذجة المضلة :

1- التحريك (Animation) : الطوبولوجيا الجيدة تعني تدفقاً صحيحاً لحلقات المضلعات حول المفاصل والأجسام، بينما الطوبولوجيا السيئة تؤدي إلى تشوهات عند ثني المفاصل أو تحريك الأجسام.

2- التظليل (Texturing & UV Unwrapping) : الطوبولوجيا المنتظمة تجعل عملية فك النموذج إلى خريطة UV أسهل وأكثر دقة.

3- التقسيم أو تنعم النموذج (Subdivision Surface) : الطوبولوجيا الجيدة تضمن أن النموذج المنعم يحافظ على شكله المطلوب ويطور تجاعيد وأنحناءات صحيحة.

ما الذي يجعل الطوبولوجيا جيدة؟

- الرباعيات المنتظمة بقدر الإمكان.
- تدفق حلقات المضلعات مع شكل الجسم (مثل دوائر حول العينين والفم).
- الكثافة المتوازنة : وجود المضلعات عندما تكون التفاصيل مطلوبة فعلاً، وليس في المناطق المسطحة.

2- نمذجة الأسطح الرياضية الناعمة NURBS

بالوقت الذي تهيمن فيه النمذجة المضلعة على معظم المشاريع البصرية، توجد تقنية أخرى قائمة على أساس رياضي متين تتفوق في مجالات محددة، هي نمذجة الأسطح الرياضية الناعمة.

NURBS هي اختصار لـ: Non-Uniform Rational B-Splines

وهي تمثيل رياضي للهندسة ثلاثية الأبعاد، تُستخدم لتمثيل وتحليل الأسطح والمنحنيات المعقدة، مما يسمح بالتحكم غير المنتظم في توزع النقاط على المنحنيات.

وهي تشبه رسم المنحيات في برامج مثل Illustrator ولكن في فضاء ثلاثي الأبعاد.

المكونات الأساسية في NURBS

أ. النقاط Control Points (CVs) : نقاط في الفضاء ثلاثي الأبعاد تتحكم في شكل المنحنى أو السطح.

ليست بالضرورة على المنحنى نفسه (على عكس النقاط في المضلعات)، تشكل ما يسمى الهيكل التحكمي Control Hull.

ب. المنحنيات Curves : وتتشكل بناءً على مواضع نقاط التحكم CVs ولديها خاصية (درجة) تحدد مدى نعومة المنحنى، ويمكن أن تكون مفتوحة أو مغلقة.

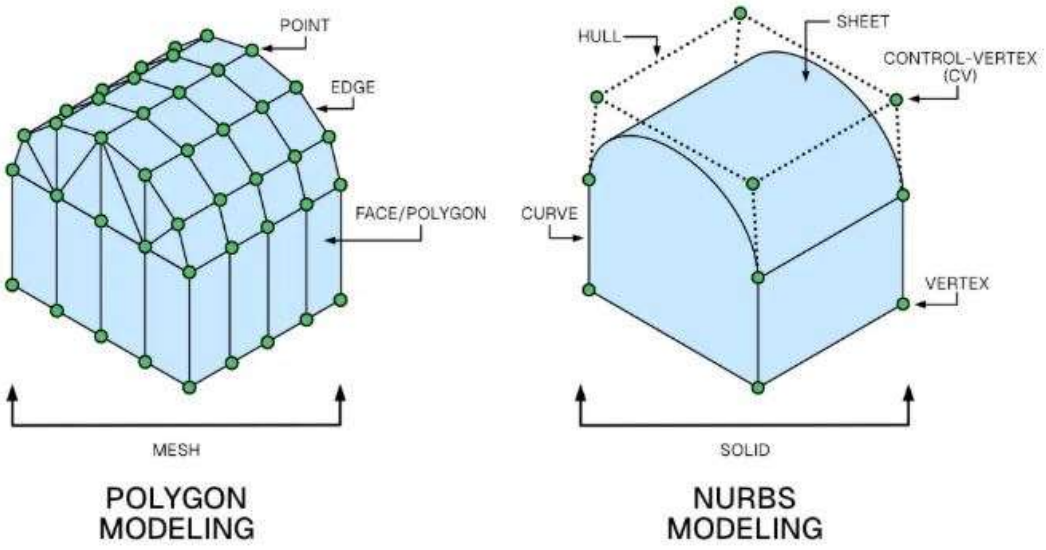
ج. الأسطح Surfaces : يتم إنشاؤها من خلال شبكة من المنحنيات في اتجاهين U و V ، تشبه شبكة مطاطية ممتدة بين منحنيات التحكم، السطح نفسه دائماً أملس ومستمر رياضياً.

المقارنة بين النمذجة المضلعة ونمذجة NURBS

المعيار	النمذجة المضلعة	النمذجة بـ NURBS

الأساس	شبكة من الوجوه المسطحة	معادلات رياضية لا أسطح منحنية
الدقة	تقريبية – تحتاج لمضلعات أكثر لدقة أعلى	دقيقة رياضياً –السطح مثالي دائماً
المرونة	مرنة جداً لأشكال معقدة وعشوائية	ممتازة للأسطح العضوية والمنتظمة
التحكم	تحكم مباشر بكل وجه وحافة	تحكم من خلال نقاط التحكم CVs

الخلاصة NURBS دقيقة رياضياً، بينما المضلعات عمليّة أكثر.



تُستخدم نمذجة NURBS في :

- التصميم الصناعي والمنتجات (سيارات، طائرات، الأجهزة الإلكترونية، الأثاث)
- هندسة العمارة (التصميمات المعمارية ذات الأسطح المنحنية ، الهياكل الزجاجية المعقدة)
- الرسوم المتحركة (إنشاء الأسطح الناعمة للشخصيات ، نمذجة المركبات والبيئات العضوية)
- التصنيع باستخدام الحاسب CAM (التحكم المباشر في آلات CNC لتصميم القوالب والقطع الهندسية)

البرامج التي تعتمد نمذجة NURBS :

- Autodesk Alias برنامج متخصص في التصميم الصناعي.
- Rhino البرنامج الأكثر شيوعاً ومرونة للتصميم المعماري والصناعي.
- Maya يحتوي على أدوات NURBS قوية بجانب المضلعات.

- Siemens NX / CATIA يستخدم في البيئات الهندسية والتصنيعية المتقدمة.
- Blender يتضمن دعم أساسي لـ NURBS.

3- نمذجة النحت الرقمي

هي عملية استخدام أدوات رقمية لمحاكاة عملية النحت التقليدية على مواد مثل الطين أو الرخام، ولكن في بيئة افتراضية ثلاثية الأبعاد.

الفرق الجوهري بينها وبين النمذجة المضلعة هو أن النمذجة المضلعة تركز على الطوبولوجيا والهيكل ، بينما يركز النحت الرقمي على الشكل والتفاصيل، إذ يمكنه العمل مع ملايين أو مليارات المضلعات، مما يسمح باضافة تفاصيل دقيقة جداً.

تستخدم نمذجة النحت الرقمي في ألعاب الفيديو، الأفلام السينمائية، الطباعة ثلاثية الأبعاد مثل المجوهرات، التماثيل، النماذج الأولية.

أدوات النحات الأساسية:

- Standard Brush: الفرشاة الأساسية للرفع والخفض.
- Clay Buildup: لبناء الطبقات بشكل طبيعي مثل الطين.
- Move Brush: لسحب وتشويه أجزاء كبيرة من النموذج.
- Smooth Brush: لتنعيم السطح وإزالة الحواف الحادة.
- Dam Standard: لإنشاء حواف حادة وتفاصيل خطية.
- Slash Brush: لإنشاء شقوق وتجاعيد عميقة.
- Inflate/Deflate: للنفخ أو التفريغ.
- Pinch Brush: لضم الحواف وجعلها أكثر حدة.

البرامج التي تعتمد نمذجة النحت الرقمي:

- ZBrush – يستخدم لإنشاء الشخصيات، المخلوقات، التفاصيل العضوية.
- Mudbox من Autodesk يستخدم لإنشاء البيئات والتفاصيل السطحية.
- Blender مناسب للمبتدئين والمشاريع الصغيرة.

طريقة النحت:

1- التشكيل البسيط (Blocking) و نحدد فيه الكتلة والحجم والتناسب الأساسي (باستخدام ZSpheres).
2- التشكيل (Sculpting) التركيز على بناء الأشكال الكبيرة (باستخدام Dynamesh للحفاظ على توزيع متساوٍ للمضلعات).
3- التفصيل (Detailing) إضافة التفاصيل المتوسطة ثم الدقيقة بزيادة كثافة المضلعات (باستخدام Alpha Brushes).
4- الاخراج (Extraction) النموذج المنحوت يكون عالي الدقة ولا يصلح للألعاب لذلك يتم إنشاء نسخة منخفضة الدقة واستخراج خرائط التشويه (Maps) مثل:

- Normal Maps: لمحاكاة التفاصيل على النموذج منخفض الدقة.
- Displacement Maps: لتشويه هندسة النموذج أثناء العرض.
- Ambient Occlusion: لإضافة ظلال طبيعية.

4- النمذجة الإجرائية

هي طريقة لإنشاء محتوى ثلاثي الأبعاد باستخدام الخوارزميات والقواعد الرياضية بدلاً من الأدوات اليدوية التقليدية.

هذه التقنية تعيد تعريف مفهوم الإبداع الرقمي، على سبيل المثال بدلاً من بناء كل مبنى في مدينة يدوياً، يمكن كتابة مجموعة من القواعد التي تبني المدينة بأكملها تلقائياً، هذا هو جوهر النمذجة الإجرائية.

المبدأ الأساسي"Write the rules, not the geometry"

تُستخدم النمذجة الإجرائية في بيئات الألعاب الضخمة (إنشاء مدن كاملة) ، التأثيرات البصرية (انفجارات) ، هندسة العمارة (تخطيط المدن، توليد واجهات معمارية معقدة)، تصميم المنتجات (كرسي مخصص بناء على وزن المستخدم).

المبادئ الأساسية للنمذجة الإجرائية:

١. المعاملات (Parameters) : تسمح بتغير النتيجة النهائية ببساطة بتعديل قيم إدخال

مثال ذلك : تغيير عدد طوابق المبنى، كثافة الغابة، تعقيد الشبكة.

٢ .العشوائية (Randomness) : استخدام البذور العشوائية (Seed) لإنشاء طبيعية متغيرة.

مثال ذلك : كل شجرة تكون فريدة بشكلها، لكنها تتبع نفس القواعد الأساسية.

٣ .التكرار (Iteration) : تطبيق نفس القاعدة بشكل متكرر لإنشاء أنظمة معقدة

مثال ذلك : فرع ينقسم إلى فروع أصغر، والتي تنقسم بدورها لفروع أخرى...

٤ .عدم التدمير (Non-Destructive) : يمكن الرجوع وتعديل أي خطوة في أي وقت وهذه ميزة فارقة عن النمذجة التقليدية.

الجدول التالي يلخص الفارق بين النمذجة التقليدية و النمذجة الإجرائية :

النمذجة التقليدية	النمذجة الإجرائية
تحكم مباشر بكل مضلع	تحكم بالخوارزميات والقواعد
عملية يدوية	عملية تلقائية
ثابتة – يصعب التعديل	ديناميكية – سهلة التعديل
مثالية للأشكال الفريدة	مثالية للأنماط والأنظمة

البرامج التي تعتمد النمذجة الإجرائية:

- Houdini يستخدم للتأثيرات البصرية، وإنشاء البيئات.
- SideFX Houdini إصدار مخصص للألعاب.
- Unreal Engine إطار عمل لإنشاء المحتوى الإجرائي.

طريقة تنفيذ النمذجة الإجرائية

- التخطيط للقواعد
  - تحليل العنصر إلى مكوناته الأساسية
  - تحديد العلاقات بين المكونات

مثال لشجرة : جذر ← جذع ← فروع رئيسية ← فروع ثانوية ← أوراق

- البناء
  - استخدام نظام العقد (Nodes) لبناء الخوارزمية
  - كل عقدة تمثل عملية أو تحويل

مثال Mesh → Subdivide → Extrude → Randomize

- التحكم والمعاملات
  - إضافة مدخلات المستخدم (Parameters)
  - ضبط نطاقات القيم المسموحة

مثال: معاملات لارتفاع المبنى من ١ إلى ١٠٠ طابق

- التصدير والتكامل
  - تحويل النتيجة إلى مضلعات ثابتة إذا لزم الأمر
  - تصدير إلى محركات الألعاب أو برامج العرض

المزايا

- كفاءة غير مسبوقة في إنشاء كميات هائلة من المحتوى
- مرونة في التعديل والتجريب، اتساق في النمط والجودة
- إمكانية إعادة الاستخدام للأنظمة

التحديات

- صعوبة التعلم
- صعوبة التحكم الدقيق في النتيجة النهائية
- صعوبة تحقيق لمسة فنية يدوية
- متطلبات حاسوبية عالية للأنظمة المعقدة

محاضرة (5) الإضاءة والألوان

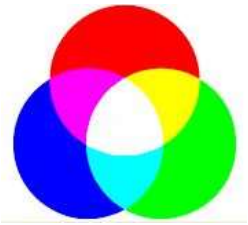
الألوان والإضاءة عنصران أساسيان في بناء المشاهد الرسومية سواء في الرسوم ثنائية البعد أو ثلاثية الأبعاد، فمن خلال التحكم في الضوء واللون، يمكن لنظام العرض أن ينقل الإحساس بالعمق، الواقعية، الملمس، والبيئة العامة للمشهد.

2- الألوان

2.1. تمثيل الألوان

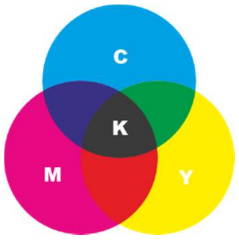
يتم تمثيل الألوان في الرسوم الحاسوبية عددياً باستخدام نماذج مختلفة، أهمها:

- نموذج RGB (Red, Green, Blue)



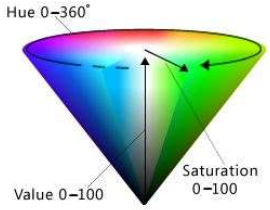
هو النموذج الأكثر استخداماً في شاشات العرض، حيث ينتج كل لون من مزيج ثلاثة ألوان (الأحمر، الأخضر، والأزرق)، يتم تمثيل اللون عادة بثلاث قيم من 0 إلى 255 في النظام الثنائي أو العشري، على سبيل المثال اللون الأبيض = 255, 255, 255.

- نموذج CMY وCMYK (Cyan, Magenta, Yellow, Black)



وهو يعبر عن امتصاص الضوء بدلاً من إشعاعه (يمكن اعتباره نموذج طرحي)، ويستخدم هذا النموذج في الطباعة، وتوجد علاقة تقابلية بينه وبين النموذج RGB كما يلي: [C = 1 − R, M = 1 − G, Y = 1 − B]

- نموذج HSV / HSL



وهو النموذج الأكثر قرباً من طريقة تفكير الإنسان في اللون:
حيث يعبر H: Hue عن تدرّج اللون ، و S: Saturation عن الإشباع،
أما V/L فهي تعبر عن السطوع Value / Lightness .

2.2. التحويل بين النماذج اللونية

كل نموذج لوني يحلّ مشكلة معينة لا يستطيع النموذج الآخر حلها بسهولة، وبالتالي التحويل بين النماذج يسمح لنا بالانتقال من عالم الأجهزة (RGB) إلى عالم الإدراك البشري (HSV/LAB) إلى عالم الطباعة (CMYK)، سنتعرض نموذجين من نماذج التحويل اللوني.

- التحويل بين RGB و CMYK

الهدف هو تحويل الألوان من النموذج الإضافي (RGB) إلى النموذج الطرحي (CMYK) .

المدخلات : R,G,B ∈ [0,1]

المخرجات: C,M,Y,K ∈ [0,1]

الخطوات الرياضية:

K = 1 − max (R,G,B)
إذا كان اللون أسود نقي
if K = 1
Return 0, 0, 0, 1
Else
نحسب القيم الطرحية
نسبة السايان (الأزرق الطرحي)
C = (1 - r - K) / (1 - K)
نسبة الماجنتا (الأحمر الطرحي)
M = (1 - g - K) / (1 - K)
نسبة الأصفر الطرحي
Y = (1 - b - K) / (1 - K)
Return C, M, Y, K

التحويل بين RGB و HSV

الهدف: تحويل الألوان من النموذج الإضافي (RGB) إلى النموذج الأسطواني (HSV – Hue, Saturation, Value).

المدخلات: R,G,B ∈ [0,1]

المخرجات: H ∈ [0, 360°) ،S ∈ [0, 1] ،V ∈ [0, 1]

الخطوات الرياضية:

حساب القيم القصوى والدنيا والفرق (Chroma)
القيمة القصوى
M = max(R, G, B)
القيمة الدنيا
m = min(R, G, B)
الفرق اللوني

```
C = M − m
حساب القيمة (Value) شدة اللون
V = M
حساب التشبع (Saturation)
if M = 0
    S = 0  اللون أسود → لا تشبع
else
    S = C / M  نسبة الفرق اللوني إلى القيمة القصوى
حساب الصبغة (Hue)
if C = 0
    H = 0  لون رمادي → صبغة غير معرفة
else
    if M = R
        H′ = (G − B) / C  mod 6
    else if M = G
        H′ = (B − R) / C + 2
    else if M = B
        H′ = (R − G) / C + 4
تحويل إلى درجات (0–360°)
    H = 60° × H′
إرجاع القيم
Return H, S, V
```

ما الذي نستفيده من الناحية التطبيقية في عملية التحويل اللوني؟

التحويل المستخدم	الفائدة الرئيسية	أمثلة واقعية ملموسة
RGB → HSV	تعديل اللون بشكل مستقل دون التأثير على الباقي	– تغيير لون السيارة في لعبة دون تغيير الإضاءة
RGB → CMYK	الطباعة الصحيحة (تجنب الألوان الفاقعة التي لا تُطبع)	– طباعة كتالوج منتجات بألوانه الأصلية
CMYK → RGB	إظهار ألوان الطباعة على الشاشة	– في الـ Photoshop View → Proof Colors
HSV → RGB	توليد تدرجات لونية سلسة	– مثل شريط التقدم progress bar

أشهر قواعد استخدام الألوان في التصميمات:

اللون هو أداة وظيفية قبل أن يكون أداة جمالية، يؤثر على جذب الانتباه، توصيل المشاعر والمزاج، تمييز العلامة التجارية، تحسين قابلية القراءة، تحفيز السلوك (Call to Action):

القاعدة	الشرح	أمثلة عن استخدامها
قاعدة 60-30-10	60% لون أساسي (خلفية)، 30% لون ثانوي، 10% لون مميز.	تصميمات Apple
قاعدة اللون الوحيد الدافئ	في واجهة باردة بالكامل، لون دافئ واحد فقط للعنصر الأهم	زر "اشترِ الآن" في Amazon
قاعدة التباين المعاكس (Complementary)	أزرق + برتقالي، أو أحمر + أخضر لجذب الانتباه الفوري	شعار FedEx بنفسجي + برتقالي
قاعدة الألوان المشابهة (Analogous)	ألوان متجاورة لإعطاء الإحساس بالهدوء والرقي	مواقع الرفاهية
قاعدة اللون الواحد أسود + أبيض فقط	هوية بصرية قوية بأقل مجهود	مثل شعار Duotone

3- الإضاءة

الإضاءة في رسوم الحاسوب هي عملية محاكاة لسلوك الفوتونات (أو الموجات الكهرومغناطيسية) من مصدر الضوء حتى وصولها إلى العين أو الكاميرا، مع الأخذ في بعين الاعتبار التفاعل مع المادة في كل نقطة على سطح المادة أو داخلها (المواد الشفافة).

يمكن بناء أدق النماذج ثلاثية الأبعاد وبملايين المضلعات وبأدق التفاصيل، لكن إذا كانت إضأته خاطئة، فسيظل يبدو النموذج أنه غير حقيقي، على النقيض، يمكن من خلال الإضاءة الصحيحة لمكعب بسيط أن يتحول هذا المكعب إلى معدن، أو رخام، أو ذهب ...

الإضاءة ليست مجرد "إضافة ضوء" على المشهد، وإنما هي محاكاة لسلوك الضوء الحقيقي، كيف ينبعث من الشمس أو المصباح، كيف يصطدم بالأسطح، كيف ينعكس أو ينكسر أو يُمتص أو يتشتت داخل المادة،

كيف يرتد آلاف المرات بين الجدران والأرض والسقف حتى يصل أخيراً إلى عين المشاهد أو عدسة الكاميرا، كل بكسل تراه على الشاشة هو في النهاية نتيجة لسؤال محدد: "كم من الضوء وصل إلى هذه النقطة من هذا الاتجاه بالضبط؟"

في عام 1986 حدد جيمس كاجيا معادلة رياضية دقيقة تقدم الإجابة على هذا السؤال، تُسمى Rendering Equation وممثلة بهذه الصيغة الرياضية:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_\Omega f_r(p, \omega_i, \omega_o)\, L_i(p, \omega_i)\, (\omega_i \cdot n)\, d\omega_i$$

الرمز	المعنى
Lo	الإشعاع الخارج (الضوء الذي يصل إلى الكاميرا)
Le	الإشعاع المنبعث (مصابيح، شمس، مواد مضيئة)
fr	انعكاس المادة
Li	الإشعاع الوارد من كل الاتجاهات
(ωi·n)	قانون كوساين لامبرت

هذه المعادلة تصف بدقة تامة كل شيء: الضوء المباشر، الظلال الناعمة، انعكاسات المرايا، تشتت الضوء داخل الجلد، لمعان المحيطات، حتى الوميض داخل الماس.

لكن المشكلة أن حلها بدقة 100% يحتاج إلى سنوات من الحساب على أقوى حاسوب موجود اليوم، لذلك، كل ما نفعله في الرسوم الحاسوبية منذ أربعين سنة هو محاولة خداع العين البشرية لنقترب قدر الإمكان من الحقيقة الفيزيائية، لكن بأقل تكلفة ممكنة.

في السبعينيات والثمانينيات، كنا نكتفي بإضاءة محلية (ضوء واحد، ظلال حادة، انعكاسات بسيطة)، وكان هذا الأمر كافياً للألعاب والرسوم المتحركة القديمة، ثم جاء عصر الإضاءة الشاملة مسبقة الحساب حيث نجهز الضوء في خرائط إضاءة، ونضع Light Probes في كل ركن، وكان ذلك كافياً لصناعة الألعاب.

ثم ظهر الـ Ray Tracing في الوقت الحقيقي عام 2018، فأصبح بإمكاننا أن نرى انعكاسات حقيقية، ظلال ناعمة، إضاءة ترتد عشرات المرات، وكل ذلك في 60 إطار في الثانية الواحدة.

اليوم، في عام 2025، ندخل عصراً جديداً هو الإضاءة العصبية، أو متعددة الأطياف، وهي الإضاءة التي لا تحتاج إلى معرفة الفيزياء مسبقاً لأن الشبكة العصبية تتعلّمها من الصور الحقيقية.

1. الإضاءة المحلية Local Illumination

فكرتها الأساسية أن نحسب الضوء في كل نقطة على السطح من مصادر الضوء المباشرة فقط ، وتتجاهل أي ضوء يأتي بعد ارتداده من سطح آخر.

في هذا النوع من الإضاءة نحسب:

- الضوء المباشر من المصابيح أو الشمس Direct Lighting
- الانعكاس اللامع Specular والمنتشر Diffuse على السطح نفسه
- الظلال الحادة فقط Hard Shadows

ما يتم تجاهله:

- الضوء المرتد (مثلا ضوء مرتد من أرض حمراء فيجعل الجدار الأبيض مائلاً للحمرة)
- الظلال الناعمة في يوم غائم
- الانعكاسات في المرايا أو الأرضيات اللامعة

من الأمثلة عن هذا النوع من الإضاءة: الألعاب التي تم تطويرها قبل عام 2005 مثل لعبة Doom

هذا النوع من الإضاءة سريع ولا يحتاج إلى موارد حاسوبية إلا أنه لا يبدو حقيقياً و تبدو المشاهد فيه كأنها بلاستيكية مسطحة لا عمق لها.

2. الإضاءة الشاملة مسبقة الحساب Precomputed Global ILLUMINATION

فكرتها الأساسية أن نحسب الإضاءة غير المباشرة مسبقاً قبل تشغيل اللعبة أو الفلم ونخزنها في خرائط أو قاعدة بيانات صغيرة، من أشهر التقنيات التي تدعم هذا النوع:

- خرائط إضاءة الأسطح Lightmaps
- نقاط الإضاءة Light Probes

في هذا النوع من الإضاءة نحسب:

- اللون المنتشر من جدار إلى آخرColor Bleeding

- الظلال الناعمة تقريبياً
- الإضاءة غير المباشرة (المرتدة) من 3 إلى 5 ارتدادات

ما يتم تجاهله:

- التغيير في الإضاءة أثناء التشغيل (مثلاً فتح باب أو كسر جدار) فإن الإضاءة لا تتغير .
- الانعكاسات اللامعة الحقيقية.
- تركيز الضوء تحت الماء أو في الكأس.

من الأمثلة عن هذا النوع من الإضاءة معظم ألعاب ما بين 2010–2022 (God of War 2018)

يتميز هذا النوع من الإضاءة بالجودة العالية والأداء الممتاز، إلا أنها لا تتفاعل مع التغييرات الديناميكية.

3. الإضاءة الشاملة في الوقت الحقيقي Real-time Global ILLUMINATION

فكرتها الأساسية أن نحسب الإضاءة غير المباشرة أثناء التشغيل، في كل إطار، مع دعم التغييرات الديناميكية، من أشهر التقنيات التي تستخدم هذا النوع من الإضاءة:

- NVIDIA RTXGI
- Unity Ray Tracing

في هذا النوع نحسب:

- إضاءة غير مباشرة ديناميكية تتغير مع الأحداث (كسر جدار، إطفاء مصباح).
- الظلال الناعمة الديناميكية
- الإنعكاسات اللامعة الحقيقية Ray-traced

ما يزال هذ االنموذج محدود بـ :

- عدد الارتدادات (عادة ما تكون بين 2–4 فقط) .
- الضوضاء لا تزال موجودة في المشاهد المعقدة.
- التأثير البصري الناتج عن تركيز الضوء بسبب الانعكاس أو الانكسار عبر الأسطح المنحنية، مثل الأنماط المتلألئة التي نراها في قاع حوض السباحة لا تزال شبه مستحيلة في الوقت الحقيقي بدقة عالية.

من الأمثلة عن هذا النوع من الإضاءة، لعبة Avatar of Pandora

تعتبر هذه الإضاءة الأقرب للواقع مع أداء مقبول مع عدد اطارت 60–120 ، إلا انها تحتاج بطاقات رسومية قوية جداً (كرت شاشة).

4. الإضاءة الشاملة غير المتحيزة Path Tracing Unbiased Global Illumination

فكرتها الأساسية أنها تتبع مسار كل فوتون بشكل عشوائي تماماً حتى يصل إلى الكاميرا، بدون أي تقريب أو خداع، وتكون النتيجة صحيحة فيزيائياً 100% عندما نأخذ عينات لانهائية باستخدام معادلة Kajiya .

يتميز هذا النوع من الإضاءة بأنه واقعي بلا حدود ويدعم الظواهر المعقدة مثل (الانعكاس أو الانكسار، انتشار الضوء تحت السطح) إلا أن معالجتها بطيئة جداً (دقائق إلى ساعات لكل إطار)، وهي غير عملية للألعاب.

من أهم الأدوات المستخدمة: Blender Cycles ، و Disney's Hyperion المستخدم في أفلام ديزني.

من الأمثلة عن هذا النوع من الإضاءة أفلام الرسوم المتحركة الحديثة مثل فلم (قلباً وقالباً Inside Out)

5. الإضاءة العصبية Neural Lighting / Neural Radiance Fields

فكرتها الأساسية هي الإنتقال من "المعادلات الفيزيائية" إلى "الدوال العصبية"

في كل تقنيات الإضاءة التقليدية من Phong إلى Path Tracing، نكتب معادلات صريحة لوصف (كيف ينعكس الضوء ، كيف يخترق المادة ، كيف يتشتت في الفضاء أو الأجسام).

أما في الإضاءة العصبية، نستبدل كل هذه المعادلات بشبكة عصبية صغيرة MLP تقوم بمهمة واحدة فقط:

نعطيها إحداثيات نقطة في الفضاء x,y,z مع اتجاه الرؤية φ وتعيد لنا :

- الكثافة density σ
- اللون أو الإشعاع RGB أو spectral
- خصائص المادة: roughness, metallic...

أي أن المشهد بأكمله والإضاءة والمادة أصبحت دالة مستمرة عصبية.

بمعنى آخر بدل من أن نحسب الإضاءة باستخدام معادلات فيزيائية، نحلها بشبكة عصبية تتعلّم الكثافة واللون من الصور الحقيقية.

5.1. من النماذج العصبية الشهيرة Neural Radiance Fields (NeRF)

فكرتها الأساسية أنها تأخذ الصور (مع أوضاع الكاميرا)، وتُدرّب شبكة عصبية صغيرة (MLP) لتتعلم الإجابة على السؤال التالي: "إذا وقفت في أي مكان في الفضاء، ونظرت في أي اتجاه، ماذا سترى بالضبط؟"

بعد التدريب، تتمكن هذه الشبكة من تمثيل المشهد بشكل ثلاثي الأبعاد: (شكل دقيق ، ألوان دقيقة، إضاءة واقعية، ظلال ناعمة، انعكاسات، شفافية، وحتى الضوء الذي يخترق الجلد أو الزجاج)، كل ذلك بدون الحاجة لمضلعات أو خرائط لونية وبدون أي تدخل يدوي.

المدخلات:

- مجموعة صور (50–300 صورة عادة)
- أوضاع الكاميرا (Camera Poses)

المخرجات:

- دالة مستمرة $f_\theta(x, y, z, \theta, \varphi) \rightarrow (r, g, b, \sigma)$
  - (x,y,z) نقطة في الفضاء
  - (θ,φ) اتجاه النظر (direction)
  - (r,g,b) اللون
  - σ الكثافة (density) أي كثافة المادة في هذه النقطة.

نقاط القوة:

1. جودة لا نهائية تقريباً ، فكلما زادت العينات زادت الدقة .
2. ظواهر فيزيائية تظهر بشكل تلقائي (انعكاسات زجاجية، ظلال ناعمة، تشتت تحت السطحي مثل الشمع، ضوء يخترق الضباب أو الدخان)
3. لا تحتاج إلا لصور عادية (ممكن صور جوال قديم)

نقاط الضعف:

1. بطيء جداً في التدريب والرسم (ساعات للتدريب)
2. لا يمكن تعديله يدوياً بسهولة (حيث لا توجد مضلعات)

3. حجم التخزين متوسط لكن الرسم بطيء
4. يعاني من الضوضاء في المناطق الفارغة

6. الرذاذ الغاوسي ثلاثي الأبعاد 3D Gaussian Splatting

وهي التقنية التي أنهت عصر NeRF الكلاسيكي عملياً ، فكرتها الأساسية بدل من أن نمثل المشهد بشبكة عصبية مستمرة (NeRF) أو مضلعات أو نقاط عادية ...

نقول أن العالم كله عبارة عن سحابة من ملايين "الكرات الغاوسية ثلاثية الأبعاد" الصغيرة جداً والمتداخلة، الشبه شفافة، و كل واحدة لها:

- مركز في الفضاء (x, y, z)
- حجم وشكل واتجاه (مصفوفة تغاير 3×3)
- لون يُخزن كـ Spherical Harmonics
- شفافية (opacity)
- خصائص مادة metallic

عندما نريد رسم الصورة، نُسقط كل هذه الكرات على الشاشة فتصبح بيضاويات ثنائية الأبعاد، ثم نرسمها طبقة طبقة من الخلف إلى الأمام مثل رذاذ الطلاء، ومن هنا جاء الاسم.

خوارزمية تدريب هذا النموذج

1. تبدأ بصور متعددة الزوايا مع أوضاع مختلفة للكاميرا .
2. تنشئ Point Cloud أولية (مثلاً 100 ألف نقطة).
3. تحول كل نقطة إلى Gaussian صغير جداً (حجم عشوائي صغير + شفافية منخفضة).
4. تبدأ بعملية التحسين (المواقع، الأحجام، الدوران، الألوان ، الشفافية) (عادة 30,000 خطوة).
5. كل 100–300 خطوة نجري تحكم كيفي في الكثافة Adaptive Density Control
    - إذا كان التدرج كبير جداً نقسم الـ Gaussian إلى اثنين Split
    - إذا كان صغير جداً أو شفافيته قريبة من الصفر، تحذفه

بعد التدريب 10–90 ثا، نحصل على مشهد ثلاثي الأبعاد بإضاءة واقعية 100% قابل للدوران 360 درجة

لماذا تفوق الرذاذ الغاوسي ثلاثي الأبعاد على NeRF ؟

#	NeRF	3D Gaussian Splatting
وقت التدريب	6–48 ساعة	8–90 ثانية فقط
سرعة الرسم	10–120 إطار في ثانية	200–3000 إطار في الثانية
الجودة البصرية	جيدة لكن فيها ضوضاء	أنظف، أكثر حدة، لا ضوضاء تقريباً
التحكم والتعديل	شبه مستحيل	سهل جداً (نحذف، نضيف، نعدل الكرات)
حجم الملف	5–100 ميجا بايت	100ميجا – 4 جيجا بايت
الإضاءة الديناميكية	ضعيف جداً	ممتاز (مع الإصدارات الحديثة)

محاضرة (6) التصور ثلاثي الأبعاد

التصور ثلاثي الأبعاد هو عملية إنشاء نماذج وصور رقمية ثلاثية الأبعاد باستخدام الكمبيوتر لتمثيل الأجسام، حيث يُعتمد على نظام إحداثيات كارتيزي (x, y, z) لتحديد مواقع النقاط. نظرياً، يُمثل الجسم الثلاثي الأبعاد عبر نماذج مثل الشبكات المضلعة (Polygon Meshes)، التي تتكون من قمم (Vertices)، حواف (Edges)، ووجوه (Faces) .

هذه النماذج تعتمد على نظرية الهندسة وتُستخدم خوارزميات لتحويل بيانات الحجوم إلى مسطحات، تُرسل هذه البيانات إلى GPU عبر Vertex Buffer Objects (VBOs) للمعالجة الفعالة، مما يقلل من الحمل على الـ CPU. ويسرع عملية الإنتاج Rendering.

بعد التمثيل، تأتي مرحلة التحولات (Transformations) التي تسمح بتغيير موقع أو شكل الجسم في الفضاء. التحولات الأساسية تشمل الإزاحة (Translation) ، الدوران (Rotation) ، والتكبير/التصغير (Scaling) ، وتعتمد على مصفوفات التحول في الجبر الخطي.

لعرض الجسم الثلاثي الأبعاد على شاشة ثنائية الأبعاد، نحتاج إلى عملية الإسقاط (Projection) هناك نوعان رئيسيان: الإسقاط المتعامد (Orthographic Projection) الذي يحافظ على التوازي دون تأثير العمق (المساقط الأمامية والعلوية والجانبية)، والإسقاط المنظوري (Perspective Projection) الذي يقلل حجم الأجسام البعيدة ليحاكي الرؤية البشرية.

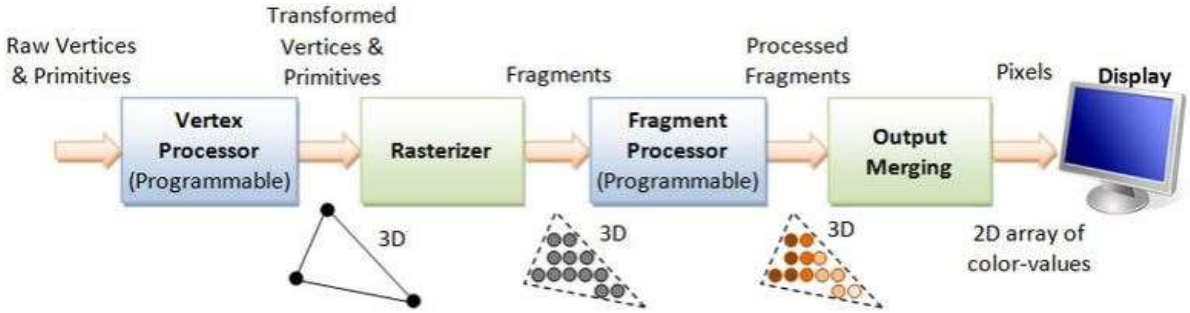
أحد الجوانب الحاسمة في التصور الثلاثي الأبعاد هو الإضاءة والتظليل (Lighting and Shading) ، الذي يضيف العمق والواقعية. نظرياً، أخيراً، تأتي تقنيات الإنتاج (Rendering Techniques) لتحويل النموذج إلى صورة نهائية، والرسم بالمسح (Rasterization) هو الأساسي، حيث يُستخدم خوارزمية Z-Buffer لإخفاء الأسطح المخفية بمقارنة قيم العمق لكل بكسل.

2- خط إنتاج الرسوم ثلاثية الأبعاد 3D Graphics Pipeline

خط إنتاج الرسوم هو سلسلة من المراحل المتتابعة التي تقوم بتحويل البيانات الخام للمشهد ثلاثي الأبعاد (القمم، النسيج، الإضاءة، الكاميرا ... إلخ) إلى صورة نهائية ثنائية الأبعاد تظهر على الشاشة، وكل هذا يتم داخل وحدة معالجة الرسوم GPU .

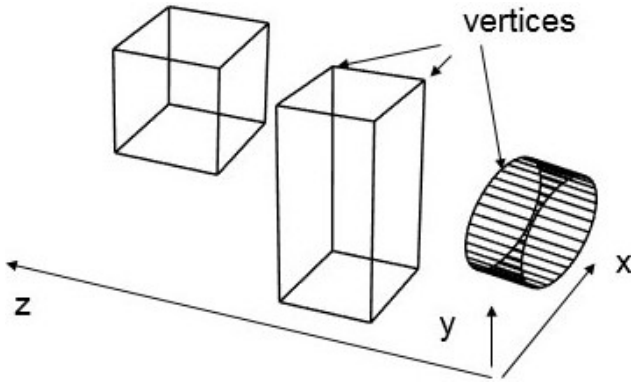
في مجال الألعاب تحديداً، هذا الخط هو عملية كاملة تحول عالماً ثلاثي الأبعاد إلى صورة بكسلات ثنائية الأبعاد (Raster Image) تراها عيناك على الشاشة بمعدل 60 أو 120 أو 240 إطاراً في الثانية.

الأساسيات التي يجب أن نعرفها أن خط الإنتاج يتكون من سلسلة من المراحل الثابتة والمرنة التي تحدث تلقائياً داخل الـGPU ، بالإضافة إلى تقنيات إضافية متقدمة يمكن إضافتها إلى هذا الخط لتحسين الأداء أو الجودة البصرية مثل Deferred Rendering ، Ray Tracing، Tessellation إلخ، باختصار خط إنتاج الرسوم هو المصنع الخفي داخل الـ GPU الذي يأخذ عالمك الثلاثي الأبعاد ويحوله إلى الصورة التي تراها على الشاشة.



المرحلة الأولى: معالجة القمم(Vertex Processing)

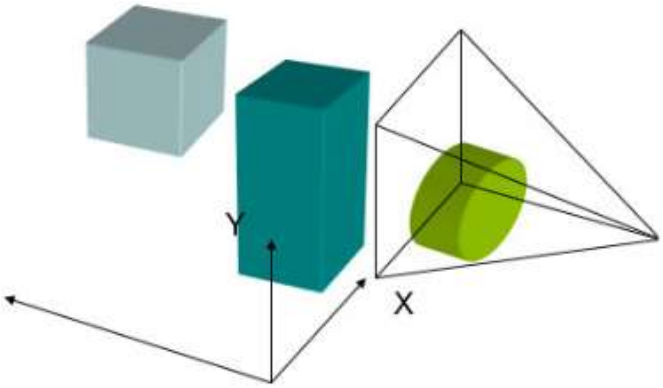
يحدد الحاسوب موقع كل جسم في المشهد ثلاثي الأبعاد ويتم ذلك بمعالجة كل قمة أو رأس (Vertex) في النماذج على حدى، ثم يربط هذه القمم مع بعضها لتكوين الوجوه (Faces) ، ويرتب هذه الوجوه داخل النموذج، بالإضافة إلى ذلك، يتم معالجة بيانات UVW Mapping كيفية لف النسيج Texture على السطح من قمة إلى قمة ومن وجه إلى وجه.



المرحلة الثانية: تحويل الرؤية (View Transform)

يحدد الحاسوب مكان الكاميرا في المشهد واتجاه نظرها نحو الأجسام ثلاثية الأبعاد، ثم يحول هذه الرؤية الثلاثية الأبعاد إلى تمثيل ثنائي الأبعاد (أي الصورة التي ستعرض على الشاشة).

يستخدم لذلك حسابات رياضية معقدة مثل مصفوفات الإسقاط Projection ثم تحول إحداثيات كل قمة من الفضاء الثلاثي إلى إحداثيات ثنائية على الشاشة.

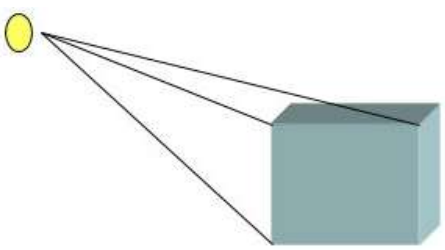


المرحلة الثالثة: القص (Scene Clipping)

يقص الحاسوب أي جزء من النماذج يقع خارج إطار الكاميرا ويتم تجاهله تماماً، وهذا يوفر في الأداء ولا نضيع وقت الـ GPU في معالجة بكسلات لن تُرى أصلاً.

المرحلة الرابعة: الإضاءة (Lighting)

يتم الآن حساب تأثير المصابيح في المشهد على وجوه النماذج والمجسمات، باستخدام تقنيات التظليل (Shading). حيث يتم حساب شدة الضوء عند كل قمة فقط، ثم يتم استكمال هذه القيم عبر الوجه كاملاً.



المرحلة الخامسة: الترسيت أو التنقيط (Rasterization)

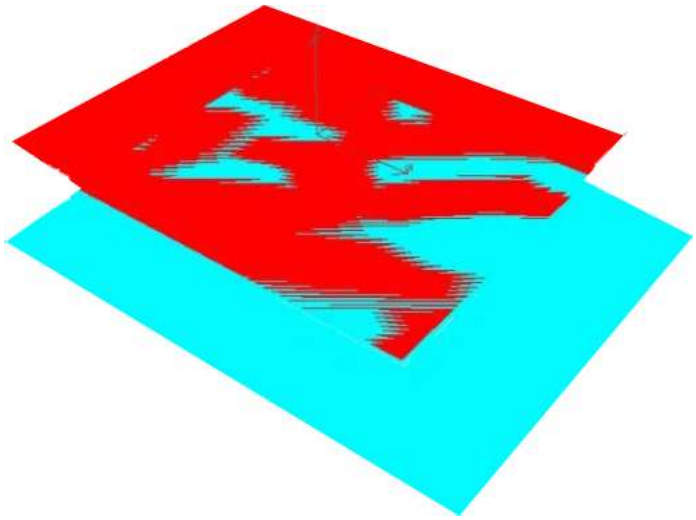
وهي المرحلة الأخيرة من خط الإنتاج، ويتم فيها تحويل الصورة ثنائية الأبعاد (المكونة من مثلثات) إلى مصفوفة من البكسلات على الشاشة، حيث يتم تحديد لون كل بكسل بناءً على النسيج (Texture) باستخدام إحداثيات UV، الإضاءة المحسوبة سابقاً ، وأي تأثيرات أخرى.

فرز العمق Z-Buffer / Z-Sort

قبل الترسيت النهائي، يجب على الحاسوب أن يعرف أي سطح أقرب إلى الكاميرا وأي سطح مخفي خلفها، يستخدم لهذا الغرض الـ Z-Buffer / Z-Sort

كل بكسل له قيمة عمق Z-value ، والبكسل الأقرب أي الذي له قيمة عمق أقل هو الذي يُرسم ، بينما يتم تُتجاهل البكسلات المخفية Z-Culling.

بدون Z-Buffer أو إذا كان دقته منخفضة يحدث تشوهات Z-Fighting بين سطحين متقاربين جداً

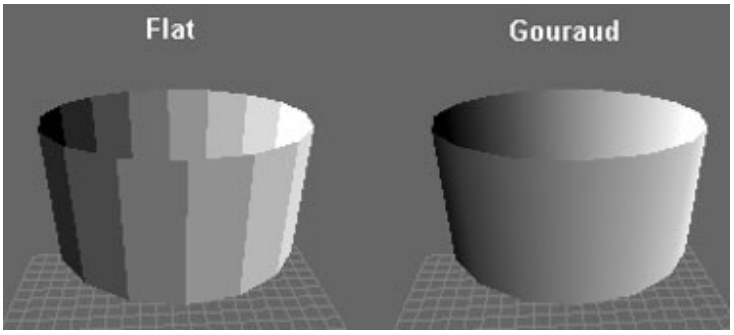


الـ Frame Buffer وتقنية الـ Double Buffering

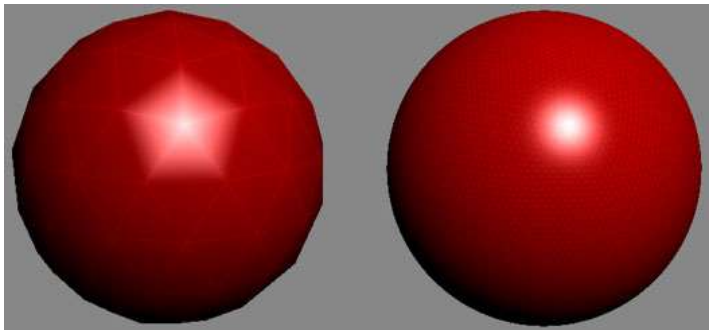
بعد انتهاء الرسم، تُرسل الصورة النهائية إلى الـ Frame Buffer لعرضها على الشاشة، تُخزن أولاً في الـ Back Buffer (مخزن خلفي)، حيث تستخدم الشاشة الـ Front Buffer للعرض، عندما ينتهي الإطار من العرض يتم "قلب" المخزنين (Swap/Flip) مما يجعل الحركة سلسة.

تقنيات التظليل (Shading Techniques)

- التظليل المسطح Flat Shading وهو الأبسط والأقل تكلفة، حيث يتم حساب لون واحد للوجه كاملاً بناءً على زاويته مع الضوء، النتيجة تبدو مضلعة وغير واقعية.
- Gouraud Shading يحسب الإضاءة عند القمم فقط ثم يستكمل اللون عبر الوجه، وهذا النوع أكثر سلاسة من Flat، لكن مع النماذج منخفضة المضلعات تظهر الانعكاسات المرآتية بشكل مسطح وغير دقيق.



- Phong Shading وهو الأفضل جودة حيث يحسب الإضاءة عبر الوجوه لكل بكسل على حدى ويعطي نتائج واقعية جداً ، حتى مع نماذج قليلة المضلعات، لكنه الأعلى تكلفة.



3- واجهات برمجة الرسوم (Graphics APIs)

الـ API هي برمجية مكتوبة بلغة منخفضة مثل C++ وتعمل كمترجم لبرامج الرسوم أشهرها Directx من مايكروسوفت و OpenGL مفتوحة المصدر، تتيح هذه الواجهات للمبرمج التحكم المباشر في وحدة معالجة الرسوم GPU، وهما العمود الفقري لأي نظام تصور ثلاثي الأبعاد (الألعاب، المحاكاة، الواقع الافتراضي، الواقع المعزز، التصميم، التصوير الطبي... إلخ).

#	DirectX	OpenGL
الشركة المطورة	Microsoft	مفتوح المصدر
المنصات المدعومة	Windows, Xbox	Windows, Linux, macOS, Android, iOS, WebGL
أحدث الإصدارات	DirectX 12 Ultimate 2020	OpenGL 4.6 2017

3.1. DirectX

أطلقت مايكروسوفت DirectX في عام 1995 لتحسين أداء الألعاب على Windows، وتتضمن عدة مكونات مثل Direct3D للرسوم ، DirectSound للصوت ، DirectInput للإدخالات.

خط أنابيب Direct3D ويتكون من المراحل التالية:

1- جمع الإدخالات Input Assembler : هذه المرحلة تقرأ البيانات من الـ Buffers الموجودة في الـ GPU وتمررها إلى الـ Vertex Shader بناءً على وصف البيانات وطريقة ترتيبها ، (الشيدرز هي برامج صغيرة جداً تعمل ضمن الـ GPU).

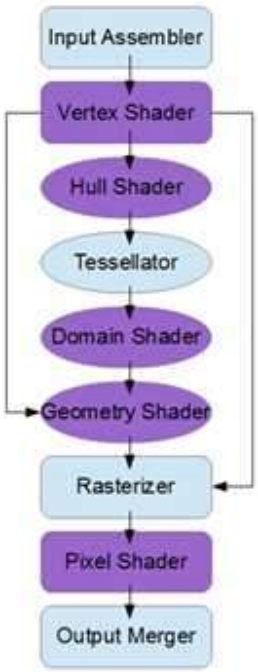
2- معالج القم Vertex Shader : هي المرحلة الوحيدة الإجبارية والقابلة للبرمجة بالكامل، تتعامل مع كل قمة (Vertex) على حدى، تحول إحداثياتها، تحسب الإضاءة الأولية، أو أي عملية أخرى تحتاجها.

3- معالج القشرة Hull Shader : وهي مرحلة اختيارية – تستخدم في التجزيء Tessellation تحول المضلعات الكبيرة البسيطة إلى رقع هندسية Patches وتحدد عدد النقاط الجديدة التي سيتم إنشاؤها.

4- المُجزِّئ Tessellator : مرحلة ثابتة غير قابلة للبرمجة تأخذ الرقع الهندسية من الـ Hull Shader وتقسمها فعلياً إلى مئات أو آلاف المضلعات الصغيرة جداً حسب عامل التجزيء.

5- معالج النطاق Domain Shader : مرحلة اختيارية – تأخذ النقاط الجديدة التي أنتجها الـ Tessellator وتحسب الموقع النهائي والإحداثيات لكل قمة جديدة.

6- معالج الهندسة Geometry Shader : مرحلة اختيارية وقابلة للبرمجة، تعمل على المضلع كاملاً مثلث أو خط أو نقطة وليس على القمم فقط. يمكنها إنشاء مضلعات جديدة ، حذف مضلعات، تحويل نقطة إلى مثلث أو أكثر .



7- Rasterizer المحول إلى بكسلات – المُرَسْتَر : وهي مرحلة ثابتة لكن يمكن ضبط إعداداتها، تحول المضلعات ثلاثية الأبعاد إلى بكسلات على الشاشة، تقرر أي بكسل داخل الشاشة وأي وجه مرئي Front-Facing وأي وجه مخفي Back-Facing أو خارج الشاشة، وترسل فقط البكسلات المرئية إلى الـ Pixel Shader وهذا يوفر في أداء عمل الـ GPU.

8- معالج البكسل Pixel Shader : المرحلة الإجبارية الثانية والقابلة للبرمجة بالكامل، تُنفذ مرة واحدة لكل بكسل مرئي، تحسب اللون النهائي، الإضاءة، النسيج ، الظلال، الانعكاسات... إلخ.

9- وحدة الدمج النهائي Output Merger : تقوم المرحلة الأخيرة باختبار العمق Depth Test (من الأقرب للكاميرا؟) ، ثم دمج

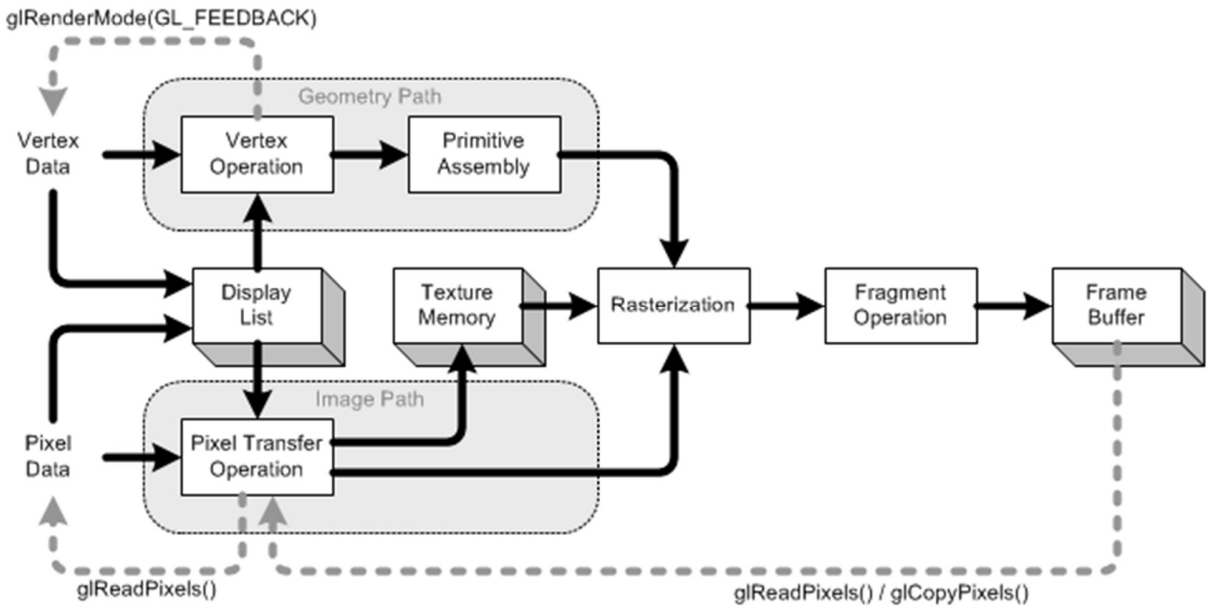
الألوان Blending إذا كانت هناك شفافية ، وكتابة اللون النهائي والعمق في الـ Render Target (الشاشة).

3.2. OpenGL (Open Graphics Library)

هو واجهة برمجة مفتوحة المصدر للرسوم الثنائية والثلاثية الأبعاد، متعددة المنصات، تم تطويرها بواسطة Khronos Group. ، بدأ OpenGL في عام 1992 كوريث لـ IRIS GL من Silicon Graphics ، وهدفه توحيد الرسوميات عبر المنصات.

خط أنابيب OpenGL يتكون من مراحل ثابتة ومراحل قابلة للبرمجة:

- إرسال البيانات Vertex Specification (Vertices, Indices)
- معالجة القمم (تحولات، إضاءة) Vertex Shader
- تجميع المضلعات Primitive Assembly
- تحويل إلى بكسلات Rasterization
- تلوين البكسلات (تظليل، نسيج) Fragment Shader
- اختبار العمق، الدمج Per-Sample Operations



يدعم OpenGL التحولات، الإسقاط، الإضاءة عبر الشيدرز ، بينما يوفر DirectX أدوات متقدمة مثل Ray Tracing.

Vulkan .3.3

Vulkan هو معيار مفتوح المصدر ومتعدد المنصات للرسوم ثلاثية الأبعاد، تم تطويره بواسطة Khronos Group ليوفر وصول عالي الكفاءة إلى GPUs الحديثة، كما يهدف إلى حل مشاكل الـ APIs القديمة مثل OpenGL التي تعتمد بشكل زائد على السواقات (Driver Overhead) كما أنهت لا تدعم للمعالجات متعددة النياسب بشكل جيد، في النهاية الهدف هو دعم أفضل للمنصات و تحقيق أداء أعلى.

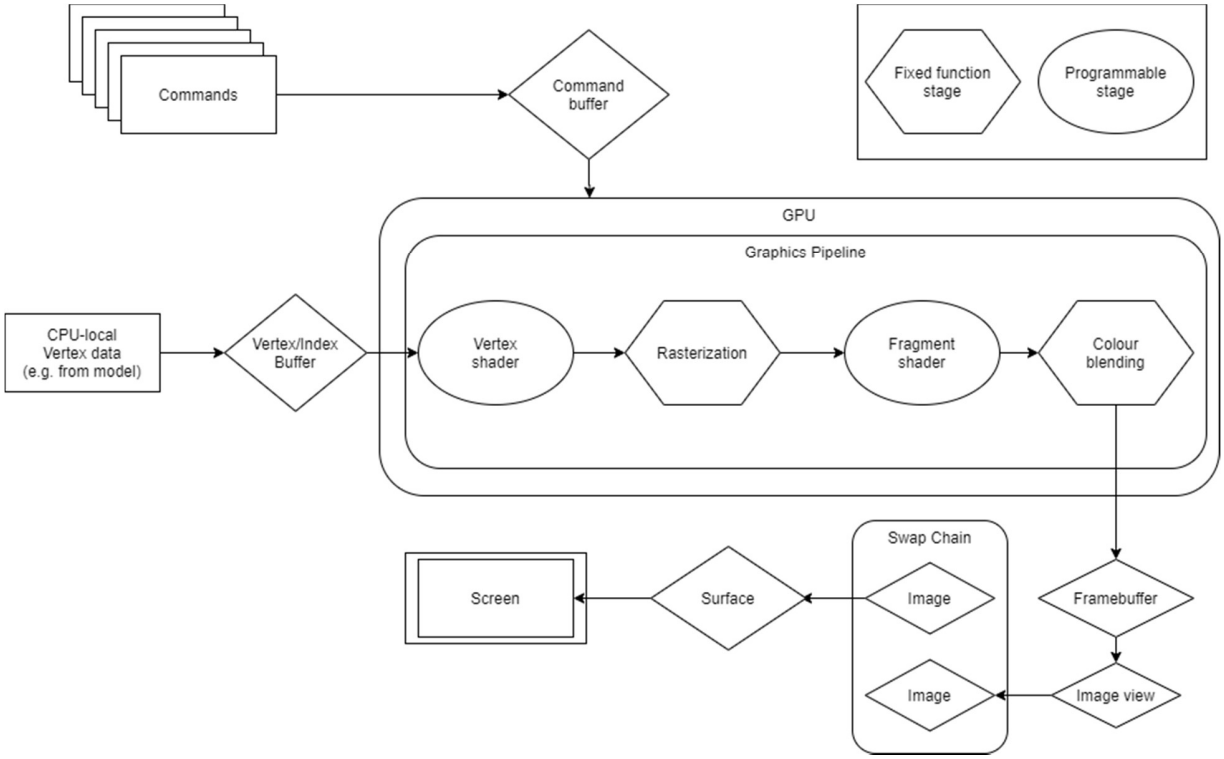
بدأ Vulkan في عام 2016 كوريث لـ OpenGL لمعالجة قيود OpenGL كما أسلفنا من خلال التركيز على تقليل الاعتماد على السوقات Driver ودعم أفضل لـ Multi-threading، في 2025، شهد Vulkan انتشاراً واسعاً وأصبح أكثر بساطة في الاستخدام مما جعله خياراً مفضلاً للمطورين الجدد.

يلعب Vulkan دوراً حاسماً في الرسوم الحديثة، خاصة في الألعاب (مثل Doom Eternal)، وفي الواقع الافتراضي والمعزز، والمحاكاة العلمية كما يدعم Ray Tracing ، مما يحسن الأداء بنسبة تصل إلى 47% في بعض الحالات، وحل في Android محل OpenGL ES لألعاب أفضل أداءً بالإضافة إلى دعم Video Encoding/Decoding لتطبيقات الوسائط المرئية.

خط أنابيب Vulkan أكثر مرونة وتحكماً مقارنة بـ OpenGL، حيث يتطلب إدارة يدوية للموارد، ومراحله الرئيسية هي:

- جمع البيانات Input Assembler (Vertices, Indices)
- معالجة القمم (تحولات) Vertex Shader
- تقسيم المضلعات Tessellation Shaders
- توليد هندسة إضافية Geometry Shader
- تحويل إلى بكسلات Rasterization
- تلوين البكسلات (إضاءة ، نسيج) Fragment Shader
- دمج الألوان والعمق Color Blending

يضيف Vulkan مراحل حسابية وخط إنتاج لـ Ray Tracing تسمح بتجميع جزئي للشيدرز لتقليل التأخير.



#	Vulkan	OpenGL	DirectX 12
المطور	Khronos Group (مفتوح المصدر)	Khronos Group (مفتوح المصدر)	Microsoft (مغلق المصدر)
المنصات المدعومة	Windows, Linux, macOS, Android, iOS	Windows, Linux, macOS, Android, iOS	Windows, Xbox
مستوى التحكم	منخفض جداً	متوسط	منخفض جداً
التركيز الرئيسي	الأداء العالي، الحسابات	الرسوم العامة	الألعاب والوسائط
سهولة التعلم	أصعب	أسهل	متوسط إلى صعب

إلى أين يتجه الوضع اليوم ؟

توقف تطوير OpenGL عند آخر إصدار لها في 2017 ، وحل Vulkan محلها في عام 2025، بينما لا تزال DirectX 12 مستخدمة على منصات Windows .

إذا لماذا ما زلنا ندرس OpenGL وDirectX رغم وجود Unity وUnreal؟

1- لفهم ما يحدث تحت الغطاء، اذ أن Unity وUnreal تستخدمان OpenGL/Vulkan/DirectX داخلياً.
2- الأداء العالي والتطبيقات المتخصصة مثل (المحاكاة العلمية، التصوير الطبي، الطيران، المحركات المخصصة) تحتاج API منخفض.
3- الواقع الافتراضي/المعزز الحديث يحتاج هذه الواجهات البرمجية.
4- الشركات الكبيرة تبني سواقاتها الخاصة فوق DirectX/Vulkan

4. الواقع الافتراضي:

الواقع الافتراضي (VR) هو بيئة ثلاثية الأبعاد تفاعلية يتم توليدها بالكامل بواسطة الحاسوب، ويتم تقديمها للمستخدم بحيث يشعر بأنه «موجود» داخلها فعلياً ، مثل السماعات الرأسية ، الشاشات كبيرة.

المتطلبات الأساسية لتجربة واقع افتراضي:

1. مجال رؤية واسع $\geq 100°$ Field of View .
2. معدل تحديث عالي ≥ 90 Hz Frame Rate .
3. زمن استجابة منخفض ≤ 20 ms Motion-to-Photon Latency .

يتضمن خط أنابيب الرسوم في الواقع الافتراضي تعديلات جذرية عن الرسوم التقليدية فهو يتطلب عرض صورتين منفصلتين لكل عين، وكل عين لها كاميرا منفصلة بإزاحة أفقية ≈ 6.3 سم.

5. الواقع المعزز

الواقع المعزز (AR) هو تقنية تدمج المعلومات الرقمية (مثل الرسومات والنماذج ثلاثية الأبعاد والصوت) في العالم الحقيقي من خلال الأجهزة مثل الهواتف الذكية والنظارات الذكية، على عكس الواقع الافتراضي الذي يستبدل الواقع بالكامل.

تلتقط الكاميرا العالم المادي المحيط وتحدد مواقع الأغراض ثلاثية الأبعاد من خلال أجهزة الاستشعار، ثم تضيف معلومات أو عناصر رقمية (مثل النصوص، الصور، أو النماذج ثلاثية الأبعاد) إلى المشهد، بشكل يتكامل مع الواقع الحقيقي، كما يمكن للمستخدمين التفاعل مع هذه العناصر الرقمية.

المتطلبات الأساسية لتجربة الواقع المعزز:

1. التتبع الدقيق والمستمر Tracking
2. فهم المشهد الحقيقي Scene Understanding
3. التوافق البصري الواقعي Lighting, Shadows, Occlusion

محاضرة (7) خوارزميات التصيير

التصيير Rendering هو عملية تحويل نموذج ثلاثي الأبعاد إلى صورة ثنائية الأبعاد لعرضها على الشاشة، مع الأخذ بعين الاعتبار (الإضاءة والظلال، المواد والأنسجة، التأثيرات البصرية، منظور الكاميرا).

التصيير عملية مهمة وأساسية في رسوم الحاسوب وخصوصاً في تطبيقات الألعاب، الأفلام، المحاكاة وتستخدم الرياضيات والفيزياء والبرمجة لإنتاج الصورة النهائية.

2- أنواع التصيير

2.1. التصيير في الوقت الحقيقي Real-time Rendering

يُعرض فيه 30-60 إطار بالثانية للموازنة ما بين الجودة والأداء، وهذا النوع مناسب للألعاب والمحاكاة التفاعلية.

2.2 التصيير غير التفاعلي Offline Rendering

يستخدم هذا النوع من التصيير لإنتاج الأفلام والصور عالية الجودة، اذ أنه يتطلب وقت معالجة طويل (دقائق إلى أيام) لكل إطار، ويحقق جودة رسومية عالية.

2.3. التصيير بالمشاهدة View-dependent rendering وهو بشكل مختصر عبارة عن التأثيرات بصرية التي تختلف بحسب موقع وزاوية الكاميرا ، وقد يكون جزءاً مكملاً في النوعين السابقين.

3- خوارزميات التصيير الأساسية:

3.1. خوارزمية مخزن العمق Z-Buffer

فكرتها الأساسية، أن كل بكسل في الصورة يحتفظ بقيمتين: (لون الشيء الذي نراه، ومسافة هذا الشيء عن الكاميرا).

مثلاً اذا كنت في غرفة مظلمة ومعك كشاف، فإن ما يُسلَط عليه ضوء الكشاف ستظهر تفاصيله، والأشياء القريبة ستحجب الأشياء البعيدة.

المشكلة التي تحلها Z-Buffer أنها تضمن ترتيب الأشياء، فإذا رسمنا إنسان قريب أولاً ثم شجرة بعيدة، سيظهر الإنسان القريب خلف الشجرة البعيدة (وهذا خطأ)، استخدام Z-Buffer يضمن ظهور الإنسان أمام الشجرة.

خطوات خوارزمية Z-Buffer

1. التحضير الأولي
   a. تهيئة مخزن العمق لكل بكسل ← نضع قيمة كبيرة جداً
   b. تهيئة مخزن اللون لكل بكسل ← نضع فيه لون الخلفية
2. معالجة كل مثلث في المشهد
   a. حساب سطح المثلث
   b. تحديد البكسلات التي يغطيها المثلث

   لكل بكسل داخل المثلث:

   أ. نحسب عمق البكسل (بُعد النقطة عن الكاميرا).

   ب. نقارن عمق البكسل مع مخزن عمق البكسل

   إذا كان (عمق البكسل < مخزن عمق البكسل):

   نحدث مخزن عمق البكسل ← نضع فيه عمق البكسل

   نحدث مخزن لون البكسل ← نضع فيه لون البكسل في المثلث

   وإلا:

   نتجاهل (أي أننا لا نجري تغيير)

3. النتيجة النهائية

   بعد معالجة كل المثلثات: سيحتوي مخزن اللون على الصورة النهائية.

يمكن تلخيص الفكرة الأساسية للخوارزمية: بأن الأقرب يغطي الأبعد، لا يهم من يرسم أولاً، لأن كل بكسل له ذاكرة خاصة به.

```
# Initialize
z_buffer = [[INFINITY for x in range(width)] for y in range(height)]
color_buffer = [[BACKGROUND for x in range(width)] for y in range(height)]

# Render all triangles
for triangle in scene.triangles:
    # Find triangle's screen bounds
    x_min = floor(min(triangle.vertices.x))
    x_max = ceil(max(triangle.vertices.x))
    y_min = floor(min(triangle.vertices.y))
    y_max = ceil(max(triangle.vertices.y))

    # Scan through bounding box
    for y in range(y_min, y_max + 1):
        for x in range(x_min, x_max + 1):
            if inside_triangle(x, y, triangle):
                # Compute depth (z-value)
                z = compute_depth(x, y, triangle)

                # Depth test
                if z < z_buffer[y][x]:
                    z_buffer[y][x] = z
                    color_buffer[y][x] = shade_pixel(x, y, triangle, z)

# Output final image
output_image(color_buffer)
```

مثال عملي: اذا كان لدينا مشهد يتضمن كرة حمراء قريبة من الكاميرا ومكعب أزرق خلف الكرة ...

بدون استخدام الـ Z–Buffer فإن رسم المكعب قبل الكرة سيؤدي لظهور المكعب واختفاء الكرة وهذا خطأ.

ما تتميز به خوارزمية Z–Buffer أنها بسيطة في التنفيذ، سريعة جداً ودقيقة تماماً، إلا أن من عيوبها التكرار، فقد ترسم نفس البكسل عدة مرات، وتحتاج لذاكرة كبيرة، ولا تدعم الشفافية.

تطبيقاتها العملية: ألعاب الفيديو، برامج التصميم، الواقع الافتراضي.

3.2. خوارزمية تتبع الأشعة Ray Tracing

فكرتها الأساسية ، هي محاكاة لحركة الضوء في العالم الحقيقي، شعاعاً بشعاع.

مثلاً اذا كنت في غرفة مظلمة وأشعلت شمعة فإن الضوء يخرج من الشمعة في كل الاتجاهات ويصطدم بالأشياء (جدران، طاولة، كرسي)، بعض الضوء يتم امتصاصه وبعضه ينعكس وبعضه ينكسر، أخيراً جزء من الضوء يعود إلى العين.

خوارزمية تتبع الأشعة، محاكاة عكسية لحركة الضوء (من العين => إلى الأشياء => إلى مصدر الضوء).

خطوات خوارزمية Ray Tracing

الخطوة 1: إرسال الأشعة لكل بكسل على الشاشة:

1. ارسم خطاً وهمياً شعاع من العين (الكاميرا)
2. عبر هذا الشعاع إلى العالم الافتراضي، اسأل: أي شيء يصطدم به هذا الشعاع أولاً؟

الخطوة 2: البحث عن الاصطدام

إذا اصطدم الشعاع بشيء:

1. احسب نقطة الاصطدام بدقة
2. اسأل: ماذا يحدث للضوء هنا؟
3. أرسل أشعة جديدة:

شعاع نحو مصدر الضوء (لون الشيء)

شعاع منعكس إذا كان لامعاً

شعاع منكسر إذا كان شفافاً

الخطوة 3: تتبع الأشعة الثانوية

لكل شعاع جديد كرر الخطوة 2

تتبع حتى يصل إلى مصدر الضوء أو يضيع في الفراغ أو يتجاوز عدد الارتدادات المسموحة

الخطوة 4: اجمع كل الألوان من:

1. الضوء المباشر من المصادر
2. الانعكاسات من الأسطح
3. الانكسارات عبر المواد
4. الظلال المناطق التي لا يصلها ضوء

```
def ray_trace(ray, depth):

    if depth > MAX_DEPTH:
        return BACKGROUND_COLOR

    # Find closest intersection
    hit_info = find_closest_intersection(ray)

    if not hit_info.hit:
        return BACKGROUND_COLOR
    # Initialize color
    color = BLACK
    # Direct illumination (shadows)
    for light in scene.lights:
        shadow_ray = create_shadow_ray(hit_info.point, light.position)

        if not is_in_shadow(shadow_ray):
            # Add light contribution
            color += calculate_lighting(hit_info, light, ray)
    # Reflections
    if hit_info.material.is_reflective:
        reflection_ray = calculate_reflection_ray(ray, hit_info)
        reflection_color = ray_trace(reflection_ray, depth + 1)
        color += hit_info.material.reflectivity * reflection_color
    # Refractions
    if hit_info.material.is_transparent:
        refraction_ray = calculate_refraction_ray(ray, hit_info)
        refraction_color = ray_trace(refraction_ray, depth + 1)
        color += hit_info.material.transparency * refraction_color

    return color
```

4. أنواع الأشعة المستخدمة في الـ Ray Tracing:

1. الشعاع الأساسي Primary Ray (يخرج من الكاميرا ويمر عبر كل بكسل)

2. أشعة الظل Shadow Rays (من نقطة الاصطدام إلى مصدر الضوء)

إذا اصطدمت بشيء قبل الضوء ← ظل ، أما إذا وصلت للضوء مباشرة ← النقطة مضاءة.

3. أشعة الانعكاس Reflection Rays (تنعكس من الأسطح اللامعة)

وهذه الأشعة تتبع القانون التالي: "زاوية السقوط = زاوية الانعكاس"

التأثير البصري للأشعة المنعكسة قد يكون (مرآة، معدن، سطح لامع)

4. أشعة الانكسار Refraction Rays (تمر عبر المواد الشفافة)

وتنكسر بحسب معامل الانكسار ، التأثير البصري للأشعة المنكسرة قد يكون (زجاج، ماء، بلورات).

5. أشعة منتشرة Diffuse Rays (تنتشر في كل الاتجاهات)

تستخدم للمواد غير اللامعة مثل الخشب والقماش وتعطي إضاءة ناعمة وظلال خفيفة.

5. المعادلات الرياضية الأساسية:

5.1. معادلة الشعاع: $P(t) = Origin + t * Direction$

حيث:

- P(t): النقطة على الشعاع عند المسافة t
- Origin: نقطة بداية الشعاع (الكاميرا)
- t: المسافة على طول الشعاع
- Direction: اتجاه الشعاع

5.2. معادلة الانعكاس: $R = I - 2 * (I \cdot N) * N$

حيث:

- R: الشعاع المنعكس بعد عملية الانعكاس
- I: الشعاع الوارد (أي الشعاع الأصلي)
- N: هو الشعاع العمودي للسطح الذي يتم الانعكاس عنه
- · : هو حاصل الضرب النقطي بينهما.

5.3. معادلة الانكسار (قانون سنل): $n_1 * \sin(\theta_1) = n_2 * \sin(\theta_2)$

حيث:

- n_1, n_2: معاملات الانكسار
- θ_1, θ_2: زوايا السقوط والانكسار

مثال عملي:

اذا كان لدينا مشهد يتضمن كرة حمراء لامعة، مكعب أزرق غير لامع ، لوح زجاجي شفاف، مصباح واحد

فإن عملية التتبع لبكسل في منتصف الصورة تكون كمايلي:

1. الشعاع يصطدم بالكرة أولاً
2. أرسل شعاع ظل إلى لمصباح: لا عوائق ← الكرة مضاءة ← اللون: أحمر + توهج المصباح
3. أرسل شعاع انعكاس (لأن الكرة لامعة): الشعاع المنعكس يصطدم بالمكعب ← لا انعكاس (لأن المكعب غير لامع) ← اللون: أزرق
4. أرسل شعاع انكسار (لأن هناك زجاج): الشعاع ينكسر عبر الزجاج ← يرى خلف الزجاج خلفية المشهد
5. النتيجة بتجميع الألوان هي: أحمر مضاء + انعكاس أزرق + انكسار خلفية

6. مشاكل الـ Ray Tracing:

6.1. البطء بسبب ضخامة العمليات (مليون شعاع × 10 ارتدادات = 10 مليون عملية)

لتجاوز هذه المشكلة يمكن تقليل عدد العينات في المناطق المتجانسة.

6.2. الضوضاء Noise بسبب قلة العينات، و عشوائية الأشعة

لتجاوز هذه المشكلة يمكن زيادة العينات، واستخدام تقنيات تخفيف الضوضاء

7. تطبيقات الـ Ray Tracing:

7.1. الأفلام والمحتوى المرئي: (أفلام ديزني ، التأثيرات المرئية في هوليوود، إعلانات السيارات)

7.2. التصميم والمحاكاة: (التصميم المعماري الإضاءة الطبيعية، تصميم السيارات، انعكاسات الطلاء تصميم المجوهرات، انكسارات الألماس)

7.3. البحث العلمي: (محاكاة انتشار الضوء في الأنسجة، دراسة خصائص المواد البصرية، تصميم العدسات والمرايا)

7.4. الألعاب الحديثة: (الإضاءة الشاملة، الظلال الناعمة، الانعكاسات الديناميكية).

8. خوارزمية تتبع المسار Path Tracing:

هي نسخة مطورة عن خوارزمية تتبع الأشعة، الفارق بينهما أن الـ Ray Tracing يتتبع شعاعاً واحداً لكل عملية ضمن مسارات محددة مسبقاً، بينما يتتبع الـ Path Tracing آلاف المسارات العشوائية ويحاكي الضوء المنتشر بشكل طبيعي وبالتالي هي أكثر واقعية، ولكنها أبطأ.

محاضرة (8) أساسيات التصيير في الرسوم المتحركة

التصيير Rendering في الرسوم المتحركة هو العملية النهائية التي تتحول فيها النماذج والحركات والإضاءة إلى مشهد مرئي متكامل يشبه الفيلم الحقيقي.

يوحد التصيير بين عناصر المشهد ويضيف لها التأثيرات البصرية، وفي النهاية يبث الحياة في الصورة.

هناك عدة أنواع للتصيير منها ما هو في الوقت الحقيقي ويستخدم في الألعاب والعروض التفاعلية ، وتصير غير تفاعلي كالمستخدم في إنتاج الأفلام والاعلانات والمشاريع عالية الجودة، بالإضافة إلى تصيير هجين يدمج بين التقنيتين.

الفهم الجيد للأساسيات (الإضاءة، المواد، الكاميرات، وتقنيات التصيير المختلفة) هو المفتاح لإنتاج رسوم متحركة عالية الجودة.

2- خط أنابيب تصيير الرسوم المتحركة

يمر خط أنابيب التصيير بعدة مراحل:
النمذجة ثلاثية الأبعاد ← التحريك ← الإضاءة ← التصيير ← التركيب ← المونتاج النهائي.

التسلسل النموذجي للتصيير يكون بحسب الخطوات التالية:

1. إعداد المشهد Scene assembly: (تجميع النماذج، وضع الكاميرات، إعداد الإضاءة).
2. المعالجة المسبقة Pre-processing: (تنظيف البيانات، تحسين النماذج، إعداد الملمس Texture).
3. التصيير Rendering: (حساب الإضاءة، تطبيق المواد، توليد الصورة).
4. المعالجة اللاحقة Post-processing (تصحيح الألوان، إضافة المؤثرات).

3- المفاهيم الأساسية المستخدمة في تصيير الرسوم المتحركة

3.1. الإضاءة Lighting

أنواع مصادر الإضاءة:

- الإضاءة الموجهة Directional Light مثل الشمس

- الإضاءة النقطية Point Light مثل المصباح
- الإضاءة المسلطة Spot Light مثل الكشاف المستخدم في المسرح
- الإضاءة المحيطة Ambient Light وهي الإضاءة العامة

للإضاءة خصائص هي: (الشدة Intensity، اللون Color، الظلال Shadows، الوهج Bloom/Glow).

3.2 المواد والملمس Materials & Textures

خصائص المادة: (القيمة الأساسية للون، مدى معدنية المادة، انعكاسية السطح، درجة الشفافية).

خصائص الملمس: (مستوى خشونة السطح، خريطة التفاصيل السطحية، مستوى الوضوح).

3.3. الكاميرات Cameras

للكاميرا خصائص وإعدادت هي: (معدل البعد البؤري Focal Length، فتحة العدسة Aperture، سرعة الغالق Shutter Speed، العمق الميداني Depth of Field).

4. التقنيات الأساسية لتصيير الرسوم المتحركة

4.1 التصيير بالمسح السطري Scanline Rendering: يعالج الصورة سطراً بسطر من الأعلى للأسفل، مع تحديد الأسطح المرئية وحساب لون كل بكسل على ذلك السطر، مما يجعله فعالاً في معالجة التفاصيل المرئية بسرعة، وهذا النوع مناسب للرسوم المتحركة.
التطبيقات التفاعلية الحديثة تستبدل هذا النوع بأساليب أحدث مثل Z-Buffering .

4.2 التصيير بالإشعاع Radiosity: يعتمد هذا النوع على حساب الإضاءة غير المباشرة، ويحاكي انتقال الضوء بين الأسطح المنتشرة بشكل واقعي، ويُظهر تفاعلات الضوء المعقدة (مثل تلوين الظلال والظلال الناعمة) بشكل مستقل عن زاوية المشاهدة، للحصول على صور ثلاثية الأبعاد أكثر واقعية، يستخدم بشكل خاص في الإضاءة الداخلية (أثناء النهار)، إلا أنه يستغرق وقتاً طويلاً.

4.3 التصيير بتتبع الأشعة Ray Tracing: يحاكي سلوك الضوء الطبيعي، دقيق ومناسب للمشاهد المعقدة إلا أنه أيضاً بطيء.

4.4 التصيير بالمسح الأمامي Forward Rendering

يتم حساب إضاءة كل بكسل لكل كائن يتم عرضه على حدى، وذلك بالمرور على الكائنات وتطبيق تأثيرات الإضاءة والمواد مباشرة عليها (الضوء ← الكائن ← البكسل)، غالباً ما يكون هناك حد لعدد الأضواء التي يمكن حسابها بدقة لكل كائن.

يستخدم هذا النوع في المشاريع التي تتطلب جودة إضاءة عالية مع عدد محدود من الأضواء للحصول على تحكم دقيق في تأثيرات الإضاءة الفردية (مثل الانعكاسات).

4.5 التصيير المؤجل Deferred Rendering

هذا النوع يفصل بين عملية تجميع معلومات الأسطح والمواد (G-Buffer) عن عمليات حساب الإضاءة، مما يزيد الكفاءة بشكل كبير، خاصة في المشاهد التي تحتوي على عدد كبير من مصادر الإضاءة، حيث يتم حساب تأثير الإضاءة لكل بكسل مرة واحدة فقط بعد تجميع المعلومات، مما يوفر موارد المعالج الرسومي (GPU) ويسمح بتصيير أضواء أكثر تعقيداً (الكائن ← معلومات السطح ← معالجة الإضاءة)، من سيئاته صعوبة معالجة الشفافية.

5. الأدوات المستخدمة في التصيير

5.1. إعدادات التصيير

1. دقة الصورة Resolution

2. معدل الإطارات Frame Rate

3. جودة العينات Sampling Quality

4. إعدادات الإضاءة Lighting Settings

5. خيارات الظلال Shadow Options

5.3 تنسيقات ملفات التصيير

1. تنسيقات ملفات النماذج ثلاثية الأبعاد

- FBX تنسيق شامل يدعم النماذج، الرسوم المتحركة، والمواد.

- OBJ يخزن الأسطح والمواد عبر ملف MTL ويستخدم للتبادل بين التطبيقات.
- gITF تنسيق حديث ومفتوح، مثالي للويب، يدعم المواد المتقدمة والرسوم المتحركة.
- DAE يدعم النماذج والمواد والأنسجة.
- STL يستخدم بشكل أساسي للطباعة ثلاثية الأبعاد.

تنسيقات ملفات الإخراج النهائي

PNG يدعم الشفافية والجودة العالية، مثالي للويب والتطبيقات التي تتطلب خلفية شفافة.

JPG/JPEG للصور الفوتوغرافية يتميز بالضغط الجيد، لكنه يفقد الجودة مع كل عملية حفظ.

TIFF جودة عالية دون ضغط، مناسب للطباعة الاحترافية.

U3D / KMZ لتضمين نماذج تفاعلية في ملفات PDF أو الخرائط (مثل Google Earth).

MP4 / MOV لتصدير مشاهد التصيير كفيديوهات أو صور متحركة.

5.3. برامج التصيير الشهيرة

- Unreal Engine التصيير بالوقت الحقيقي ويستخدم في الألعاب والأفلام
- RenderMan تصيير غير تفاعلي مستخدم في أفلام بيكسار
- Arnold تصيير غير تفاعلي مستخدم في الأفلام والإعلانات
- V-Ray تصيير هجين مستخدم في الأفلام والألعاب
- Blender Cycles تصيير هجين ويستخدم في مشاريع متنوعة

6. مشاكل التصيير والحلول العملية

6.1. زمن التصيير

الحلول: يمكن استخدام Render Farms، تقسيم المشهد ، التحكم بإعدادات الجودة

6.2. مشاكل الذاكرة:

الحلول: تخفيض دقة الأسطح ، تنظيف المشهد من العناصر غير الضرورية، التصيير على مراحل

7. الاتجاهات الحديثة في التصيير

7.1. التصيير في الوقت الحقيقي باستخدام محركات Unreal Engine، Unity

ويستخدم لإنشاء صور ومقاطع فيديو تفاعلية وبسرعة عالية (مثل الألعاب، المحاكاة، الواقع المعزز)، حيث تتولى هذه المحركات عرض بيئات ثلاثية الأبعاد بشكل فوري بناءً على إدخالات المستخدم، مع التركيز على دمج الواقع الافتراضي مع الحقيقي، وإدارة الإضاءة والظلال والكاميرات الافتراضية لإنتاج مشاهد بصرية واقعية وبالوقت الحقيقي.

كيف تعمل هذه المحركات؟

- يستقبل المحرك مدخلات من المستخدم (حركة، نقرات) أو من مستشعرات (في الواقع المعزز).
- يقوم المحرك بتحديث العالم الافتراضي بناءً على هذه المدخلات.
- تقوم وحدة معالجة الرسومات (GPU) برسم المشهد بالكامل (المجسمات، الإضاءة، الظلال، المؤثرات) لكل إطار، بمعدل تحديث مرتفع.
- يتم عرض الإطار الناتج على الشاشة، مما يعطي إحساسًا بالواقعية والتفاعل الفوري.

7.2. التصيير السحابي باستخدام خدمات مثل AWS، Google Cloud

بدلاً من استخدام أجهزة محلية يتم استخدام الخدمات السحابية لتنفيذ عمليات الرسوم ثلاثية الأبعاد وتحريكها وتوليد المؤثرات البصرية، مما يوفر في التكاليف، ويعطي مرونة في استخدام هذه الخدمات.

كيف يعمل التصيير السحابي؟

- يتم نقل ملفات المشاريع إلى مراكز مزودي الخدمات السحابية.
- عند الحاجة للتصيير، يتم استئجار عدد كبير من الخوادم السحابية القوية (CPU/GPU) حسب حجم المشروع، ويتم إنشاء مزارع تصيير افتراضية Render Farms.
- يتم توزيع مهمة التصيير على هذه الخوادم لتقليل زمن التنفيذ بشكل كبير.
- بعد الانتهاء، يتم تنزيل ملفات الفيديو أو الصور النهائية إلى جهاز المستخدم.

7.3. التصيير بالذكاء الاصطناعي

هو استخدام خوارزميات الذكاء الاصطناعي والتعلم الآلي لتسريع عملية التصيير، تحسين جودتها، أو حتى توليد صور ومشاهد كاملة دون استخدام طرق التصيير التقليدية.

7.3.1. تقنيات التصيير باستخدام الذكاء الاصطناعي:

1. الشبكات العصبية التلافيفية (CNNs):

```
# مثال: رفع دقة الصورة
class SuperResolutionCNN:
    def upscale(self, low_res_image):
        # مراحل المعالجة
        features = extract_features(low_res_image)       # استخراج الميزات
        details = predict_missing_details(features)      # تنبؤ بالتفاصيل
        high_res = reconstruct_image(features, details) # بناء الصورة عالية الدقة
        return high_res  # 4K بدقة فوتوغرافية
```

2. المحولات (Transformers):

```
# مثال: توليد صور من النصوص
class TextToImageAI:
    def generate_from_text(self, prompt):
        # مثال: "قطة تلبس قبعة في الفضاء"
        tokens = tokenize(prompt)           # تحويل النص إلى رموز
        latent = transformer_encode(tokens) # تمثيل في الفضاء الكامن
        image = transformer_decode(latent)  # فك التشفير إلى صورة
        return image  # صورة واقعية تماماً
```

3. الشبكات الخصومية (GANs):

```
# مثال: تصيير واقعي
class GAN_Renderer:
    def __init__(self):
        self.generator = NeuralNetwork()      # مولِّد
        self.discriminator = NeuralNetwork()  # مميِّز
    def render(self, scene_description):
        # المولد: ينتج صوراً
        fake_image = self.generator(scene_description)
        # المميز: يحاول معرفة إذا كانت حقيقية
        is_real = self.discriminator(fake_image)
        # التدريب: المولد يحاول خداع المميز ➔ النتيجة: صور واقعية أكثر فأكثر
```

7.3.2. مثال تطبيقي للتصيير باستخدام الذكاء الاصطناعي

استخدام الـ Ray Tracing مع الألعاب بطيء، فما هو الحل؟

يمكن تصيير اللعبة بدقة منخفضة (540 بكسل) ثم باستخدام نموذج DLSS

Deep Learning Super Sampling – NVIDIA يمكن رفع الدقة إلى (4000 بكسل)، يتم تدريب النموذج على آلاف اللحظات من الألعاب، فيتعلم النموذج إضافة التفاصيل المفقودة، ويتمكن من رفع جودة التصيير.

مثال لتصيير ألعاب مسرع بالذكاء الاصطناعي

```
class GameRendererWithAI:
    def render_frame(self, game_state):
        # الجزء 1: التصيير التقليدي السريع
        low_res_frame = traditional_render(game_state, resolution="540p")
        # الجزء 2: التحسين بالذكاء الاصطناعي
        ai_enhanced_frame = ai_upscale(low_res_frame, target="4K")
        # الجزء 3: إضافة تأثيرات بالذكاء الاصطناعي
        if enable_ray_tracing:
        # بدلاً من Ray Tracing بشكل كامل
            predicted_reflections = ai_predict_reflections(game_state)
            ai_enhanced_frame = add_reflections(ai_enhanced_frame,
predicted_reflections)
        return ai_enhanced_frame  # Ray Tracing (4k)
```