عرض مشروع جامعي: التحكم المتقدم في رسومات OpenGL

مقدمة: الهدف من الكود والمشكلة التي يعالجها

يهدف هذا المشروع إلى استعراض قدرات مكتبة OpenGL في رسم الأشكال الهندسية والتحكم بها ديناميكيًا. يركز الكود على معالجة مشكلة عرض الأشكال ثلاثية الأبعاد مع إمكانيات متقدمة مثل تدوير الكائنات، وتطبيق الألوان والشفافية، والتحكم في وضعية العرض (مليء أو شبكي).

المشكلة الأساسية التي يعالجها الكود هي كيفية تمثيل كائنات معقدة (مثل الأرقام) في بيئة OpenGL، وتطبيق تحولات ديناميكية عليها، مع الحفاظ على ترتيب العرض الصحيح للكائنات المتداخلة.

هيكلية الحل (Logic Flow)

يتكون الحل من عدة مراحل متسلسلة لضمان عرض رسومي فعال:

- **التهيئة:** إعداد مكتبات GLFW و GLEW، وإنشاء نافذة العرض، وتحديد أبعادها.
- **تعريف الشيدرات (Shaders):** كتابة شيدر الرؤوس (Vertex Shader) وشيدر الأجزاء (Fragment Shader) لمعالجة بيانات الرؤوس وتحديد ألوان البكسلات.
- **إعداد البيانات:** تعريف إحداثيات الرؤوس والألوان المرتبطة بها للأشكال المراد رسمها (الأرقام والعقارب).
- **تكوين المخازن المؤقتة (Buffers):** إنشاء VAO و VBO لتخزين بيانات الرؤوس على بطاقة الرسوميات.
- **حلقة الرسم الرئيسية (Render Loop):** حلقة مستمرة تقوم بمعالجة المدخلات، وتحديث حالة المشهد (مثل زاوية الدوران)، ورسم الكائنات في كل إطار.
- **التحكم الديناميكي:** استخدام المتغيرات الموحدة (Uniforms) في الشيدرات للتحكم في الدوران والشفافية من الكود الرئيسي.

تحليل الدوال (Code Breakdown)

Vertex Shader

```
const char* vertexShaderSource = "#version 330 core\\n" "layout (location = 0) in vec3 aPos;\\n" "layout (location = 1) in vec3 aColor;\\n" "out vec3 ourColor;\\n" "uniform float uAngle;\\n" "void main()\\n" "{\\n" " float s = sin(uAngle);\\n" " float c = cos(uAngle);\\n" " vec3 rotatedPos = vec3(\\n" " aPos.x * c - aPos.y * s,\\n" " aPos.x * s + aPos.y * c,\\n" " aPos.z\\n" " );\\n" " gl_Position = vec4(rotatedPos, 1.0); \\n" " ourColor = aColor;\\n" "}\\0";
```

- **aPos (location 0):** يستقبل إحداثيات الرأس (X, Y, Z).
- **aColor (location 1):** يستقبل اللون (R, G, B) لكل رأس.
- **ourColor:** يمرر اللون إلى Fragment Shader.
- **uAngle:** متغير موحد يستقبل زاوية الدوران من الكود الرئيسي.
- **منطق الدوران:** يتم تطبيق معادلة دوران حول محور Z على aPos لإنشاء rotatedPos.
- **gl_Position:** يحدد الموقع النهائي للرأس بعد الدوران.

Fragment Shader

```
const char* fragmentShaderSource = "#version 330 core\\n" "out vec4 FragColor;\\n" "in vec3 ourColor;\\n" "uniform float uAlpha;\\n" "void main()\\n" "{\\n" " FragColor = vec4(ourColor, uAlpha);\\n" "}\\n\\0";
```

- **ourColor:** يستقبل اللون من Vertex Shader.
- **uAlpha:** متغير موحد يتحكم في قيمة الشفافية (Alpha) من الكود الرئيسي.
- **FragColor:** يحدد اللون النهائي للبكسل، مع تطبيق قيمة الشفافية.

تحديث البيانات (المثلثات)

```
float vertices[] = { // ... بيانات الرؤوس ... };
```

- تم تحديث مصفوفة vertices لتشمل إحداثيات (X, Y, Z) وقيم الألوان (R, G, B) لكل رأس.
- يتم تمثيل الأرقام (1, 2, 3, 6, 9) كسلسلة من المثلثات، مما يسمح بتلوينها بشكل فردي.
- تم إضافة عقرب ثوانٍ رفيع كزوج من المثلثات، مع تحديد قيمة Z مختلفة لضمان ظهوره فوق الأرقام.

تفعيل قدرات OpenGL المتقدمة

```
glEnable(GL_DEPTH_TEST);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

- **GL_DEPTH_TEST:** يضمن عرض الكائنات بشكل صحيح بناءً على عمقها، مما يمنع تداخل الأشكال.
- **GL_BLEND:** يتيح دمج الألوان مع الأخذ في الاعتبار قيمة الشفافية (Alpha).
- **glBlendFunc:** تحدد كيفية دمج الألوان، حيث يتم استخدام GL_SRC_ALPHA و GL_ONE_MINUS_SRC_ALPHA للشفافية القياسية.

المخرجات (Execution & Outputs)

عند تشغيل الكود، يتوقع المستخدم رؤية نافذة GLFW تعرض:

- **أرقام الساعة:** الأرقام 1، 2، 3، 6، 9 مرسومة على واجهة ثلاثية الأبعاد.
- **عقرب الثواني:** عقرب رفيع يدور باستمرار حول مركز الشاشة، مما يوضح تطبيق التحولات الديناميكية.
- **الشفافية:** يمكن ملاحظة تأثير الشفافية على الكائنات، خاصة إذا كانت هناك كائنات متداخلة.
- **التحكم في وضعية الرسم:** بالضغط على مفتاح SPACE، يمكن للمستخدم التبديل بين وضعية الرسم الشبكي (Wireframe) ووضعية التعبئة الصلبة (Solid Fill).
- **اختبار العمق:** تظهر الكائنات البعيدة خلف الكائنات القريبة بشكل صحيح، مما يضيف عمقًا للمشهد.

الخاتمة: التحديات التي واجهتني وكيف قمت بحلها

تحدي 1: تمثيل الأرقام المعقدة

كان التحدي الأول هو كيفية تمثيل الأرقام (مثل 2 و 3 و 6 و 9) باستخدام المثلثات في OpenGL. تطلبت هذه العملية تقسيم كل رقم إلى مجموعة من المستطيلات، وكل مستطيل إلى مثلثين، ثم تحديد إحداثيات دقيقة لكل رأس.

الحل: تم استخدام أداة رسم بياني لتحديد الإحداثيات بدقة، ثم تم تجميعها في مصفوفة vertices مع الألوان المناسبة.

تحدي 2: تطبيق الدوران والشفافية

تطبيق الدوران على عقرب الثواني والتحكم في الشفافية للكائنات المختلفة كان يتطلب فهمًا عميقًا للشيدرات والمتغيرات الموحدة.

الحل: تم تعديل Vertex Shader لاستقبال زاوية الدوران (uAngle) وتطبيقها على إحداثيات الرؤوس. كما تم تعديل Fragment Shader لاستقبال قيمة الشفافية (uAlpha) وتطبيقها على اللون النهائي. تم تفعيل GL_DEPTH_TEST و GL_BLEND في الكود الرئيسي لضمان العرض الصحيح.

تحدي 3: التحكم في وضعية الرسم

إضافة القدرة على التبديل بين وضعية الرسم الشبكي ووضعية التعبئة الصلبة كان يتطلب معالجة مدخلات لوحة المفاتيح.

الحل: تم استخدام دالة processInput للكشف عن ضغط مفتاح SPACE، ثم تم استخدام glPolygonMode للتبديل بين GL_LINE و GL_FILL.

Made with GAMMA