PATH TRACING IRRADIANCIA - VIABILIDAD

Protocolo de Verificación

Condición de éxito

Ejecutar un shader en vista PT, que lance un primer rayo (muestreando el semihemisferio superior) y reutilice el FPathTracingKernel del Path Tracing para obtener la radiancia de los rebotes.

Esquema MECE

- **Accesos/Recursos**: permisos, API, Kernel.
- **Viabilidad técnica**: arquitectura del motor, compatibilidad, dependencias.
- **Viabilidad práctica:** tiempo, coste, rendimiento, soporte en build.
- **Riesgo externo**: cambios de versión, políticas.

Condiciones de fracaso

Accesos/Recursos:

- ¿Tengo acceso desde el plugin a todas las funciones shader necesarias de Path Tracing? FPathTracingKernel, Payload, FPathState...
    - *Sí, al igual que las funciones de RayTacing. Realizar test mínimo.*

Viabilidad técnica:

- ¿Puedo engancharme a alguna parte del pipeline de renderizado de Path Tracing?
    - *Sí, me puedo enganchar en ViewSceneExtension.*
- Si me engancho en ese momento, ¿Está todo lo necesario ya renderizado?
    - *Sí, ya se ha renderizado todo lo necesario. Incluso hay opciones de engancharse posterior al tonemap.*
- ¿Puedo engancharme desde mi plugin de forma correcta?
    - *En principio sí parece, pero no hay confirmación. Realizar test mínimo.*
- ¿Es viable el shader a nivel GPU?
  - *Supuestamente sí, según está actualmente. El kernel se ejecuta en un loop: no inicia proceso nuevo de GPU. Realizar tests para comprobar.*

- ¿Tengo que usar alguna función que no esté expuesta en RENDERER_API? En caso afirmativo, ¿Puedo usarla? En caso negativo, ¿Hay alternativas?
  - *Sí. Sí. El propio código IrradiancePass.cpp ya lleva en él funciones que no expone RENDERER_API. No debería haber problemas si Build.cs es correcto.*

Viabilidad práctica:

- ¿Es demasiado coste de rendimiento aplicar esta técnica en comparación con el RT normal? ¿Merece la pena coste/oportunidad?

Riesgo externo: ya asumido que una actualización del Engine puede dejar obsoleto el plugin. Al incluir internals -> no puede salir al Marketplace. Aún así, hay formas de "engañar" al compilador con las rutas de los includes, y duplicar los internos. Se puede investigar.

Estrategias viables

1. Seguir con el shader planeado, y lanzarlo desde *SceneViewExtension*. Seguir modelo de shader debug (en PathTracing.cpp) que implementa un shader mínimo de Path Tracing, e ir implementando poco a poco todo el código.
2. Si no funcionara PT de ninguna manera: RT desde el *viewport* de Path Tracing, para aprovechar las físicas raytraceables de atmósfera, nubes... RT desde el pipeline clásico requiere modelar atmósfera y nubes (sacar "texturas" de cielo y nubes, a grosso modo).

En ambos casos, asegurar convergencia en samples/frames.

Plan

1. Probar el *viewport* de Path Tracing y asegurarse de que funciona bien.
2. Revisar código fuente (Especialmente clase ray-gen debug, PT.cpp):
   1. Cómo renderiza: ¿usa funciones PT? ¿las puedo incluir?
   2. Cómo monta el PSO + SBT.
   3. Cómo hace el binding.
   4. Parámetros estáticos (UBs) y necesarios.
3. Migrar el shader actual. Lanzarlo desde *SceneViewExtension*, y no desde *PostOpaque*.
4. Probar shader mínimo de Path Tracing. Fijarse en shader debug de PathTracing.cpp.
5. Incorporar librerías de Path Tracing, y ver qué UBs y parámetros necesitan pasar sí o sí, y detalles:
   1. Si necesita pipeline distinto.
   2. Si es necesario construir con funciones de PathTracing.h/.cpp.
6. Incorporar código completo. Aprovechar librerías ya existentes.

Tests Mínimos

- Funcionamiento correcto del viewport de PT.
- Pase RDG básico en viewport PT.
- Implementación de shader mínimo.
  - Incorporación de payload PT.
  - Incorporación de librerías PT en el shader.
  - Incorporación de funciones PT en shader.
  - Incorporación de miss/hit default en C++.

Notas Clave

- El pipeline del PT no es el mismo que el de renderizado normal. El renderizado PT sustituye al renderizado clásico.
- Es imposible implementar PT en el pipeline clásico, sin crear una copia del mismo o modificar el engine. No coincidirían los buffers de PT.
- Por este mismo motivo, no funcionan los hooks en *PostOpaque*.
- Para activar la atmósfera y nubes -> Activar *Reference Atmosphere*.
- En PT, aparece mucho *GPULightMass* pero porque comparten gran parte del mismo núcleo. En teoría, no debería preocuparme por ello.

Conclusión

En principio sí parece viable. Verificar código fuente, ejecutar tests mínimos y reconsiderar según los resultados.