BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Animación por computador

NRC: 12183 | OTOÑO 2023

Práctica 1 - Preparación del entorno de animación utilizando Python.

16/10/23

DOCENTE: M.C.C GUSTAVO EMILIO MENDOZA OLGUIN

MORENO ORTEGA MEREDITH MADAI

- 201940745

Instrucciones

Implementar las clases Object3d, Scene, Group, Camera, Mesh, Geometry y Material, junto con sus extensiones (secciones 4.2, 4.3 y 4.4 del libro de Stemkovsky y Pascale). Si no ha utilizado el framework previamente, tendrá que implementarlo desde la presentación en la unidad 2. Evidenciar la salida del ejemplo 4.6 modificado con el extra del movement rig implementado, cambiando el cubo presentado por otra forma de las implementadas.

Construir el diagrama de clases de la solución presentada.

Documentar las funciones de OPENGL utilizadas en la definición de las clases que forman el proyecto

Con el aprendizaje obtenido, responder la siguiente pregunta: ¿Por qué es necesario extender las clases geometry y material?

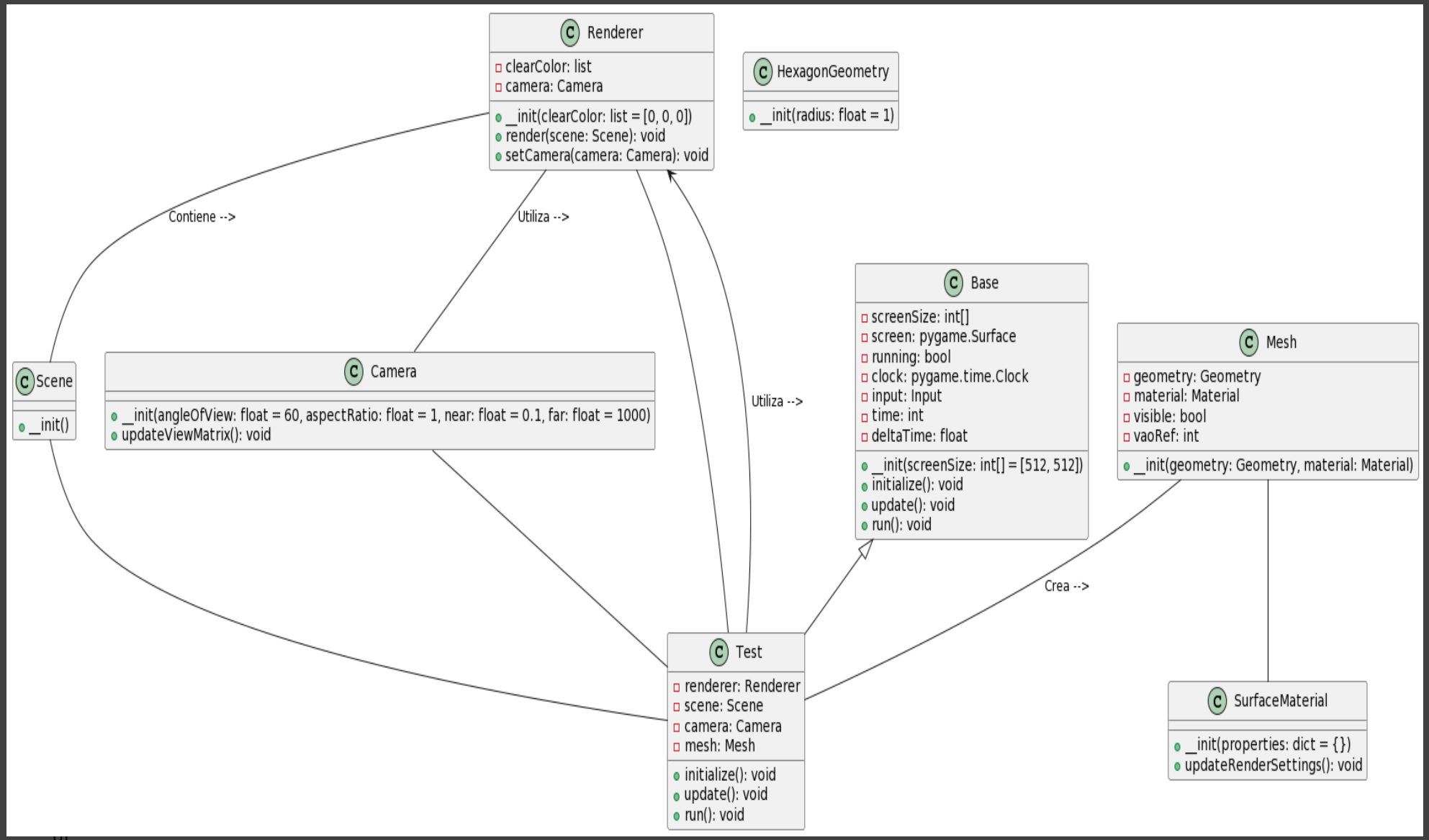
Diagrama de clases para test hexágono.

Código en PlantUml:

```
@startuml
class Base {
  - screenSize: int[]
  - screen: pygame.Surface
  - running: bool
  - clock: pygame.time.Clock
  - input: Input
  - time: int
  - deltaTime: float
  + __init(screenSize: int[] = [512, 512])
  + initialize(): void
  + update(): void
  + run(): void
}
```

```
class Renderer {
  - clearColor: list
  - camera: Camera
  + __init(clearColor: list = [0, 0, 0])
  + render(scene: Scene): void
  + setCamera(camera: Camera): void
}

class Scene {
  + __init()
}

class Camera {
  + __init(angleOfView: float = 60, aspectRatio: float = 1, near: float = 0.1, far: float = 1000)
  + updateViewMatrix(): void
}

class Mesh {
  - geometry: Geometry
  - material: Material
  - visible: bool
  - vaoRef: int
  + __init(geometry: Geometry, material: Material)
}

class HexagonGeometry {
  + __init(radius: float = 1)
}
```

```
class SurfaceMaterial {
  + __init(properties: dict = {})
  + updateRenderSettings(): void
}

class Test {
  - renderer: Renderer
  - scene: Scene
  - camera: Camera
  - mesh: Mesh
  + initialize(): void
  + update(): void
  + run(): void
}

Base <|-- Test
Renderer -- Test
Scene -- Test
Camera -- Test
Mesh -- Test: Crea -->
Mesh -- SurfaceMaterial
Renderer -- Scene: Contiene -->
Renderer -- Camera: Utiliza -->
Test -up-> Renderer: Utiliza -->
@enduml
```


Renderer
clearColor: list
camera: Camera
__init(clearColor: list = [0, 0, 0])
render(scene: Scene): void
setCamera(camera: Camera): void
HexagonGeometry
__init(radius: float = 1)
Contiene -->
Utiliza -->
Utiliza -->
Base
screenSize: int[]
screen: pygame.Surface
running: bool
clock: pygame.time.Clock
input: Input
time: int
deltaTime: float
__init(screenSize: int[] = [512, 512])
initialize(): void
update(): void
run(): void
Mesh
geometry: Geometry
material: Material
visible: bool
vaoRef: int
__init(geometry: Geometry, material: Material)
Scene
__init()
Camera
__init(angleOfView: float = 60, aspectRatio: float = 1, near: float = 0.1, far: float = 1000)
updateViewMatrix(): void
Crea -->
Test
renderer: Renderer
scene: Scene
camera: Camera
mesh: Mesh
initialize(): void
update(): void
run(): void
SurfaceMaterial
__init(properties: dict = {})
updateRenderSettings(): void

¿Por qué es necesario extender las clases Geometry y Material?

La extensión de las clases Geometry y Material es fundamental para la funcionalidad de un framework de gráficos 3D.

1. Adaptación de la geometría: Cada objeto en un entorno 3D tiene su propia forma y atributos geométricos específicos, como posiciones de vértices, colores y otros datos relacionados. Al extender la clase Geometry, se pueden crear subclases que representen formas específicas, como polígonos, cilindros o esferas. Esto permite a los desarrolladores definir geometrías personalizadas con propiedades únicas y, al mismo tiempo, mantener una estructura coherente para la manipulación de geometría en el framework.

2. Personalización del material: La apariencia visual de los objetos en una escena 3D se logra a través de sus materiales. Extender la clase Material permite definir cómo se aplican sombreados, texturas y propiedades visuales a los objetos. Esto es crucial para lograr efectos visuales realistas o estilizados en una aplicación gráfica. Al extender la clase Material, es posible adaptar los materiales a las necesidades específicas de la aplicación, lo que incluye la definición de programas de sombreado personalizados, configuración de uniformes y ajustes de renderización.

3. Flexibilidad y reutilización: Al separar geometría y material en clases independientes, se promueve la flexibilidad y la reutilización de código. Las geometrías definidas en subclases de Geometry pueden utilizarse en múltiples objetos de malla, cada uno con su propio material personalizado. Del mismo modo, varios objetos de malla pueden compartir la misma geometría y diferir solo en el material, lo que reduce la redundancia de datos y promueve una estructura modular.

4. Mantenimiento y escalabilidad: La extensión de estas clases permite un mantenimiento más eficiente del código. Si se necesitan cambios en la representación geométrica o en la apariencia visual de los objetos, es posible realizar modificaciones en las subclases de Geometry y Material sin afectar el conjunto completo del framework. Esto es esencial para aplicaciones 3D escalables y de gran envergadura.

La extensión de las clases Geometry y Material es esencial para personalizar la geometría y el aspecto visual de los objetos en un entorno 3D, lo que brinda flexibilidad, reutilización y mantenimiento eficiente del código en un framework de gráficos tridimensionales.

Link de video ejecución:

https://youtu.be/e4w6p0t-hP0?si=SBool88Ef6lJ_yH9