TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE OAXACA

SCC1010 – 6V1

Graficación

Ramos Acosta Yamileth

López Cruz Perla

Ingeniería en sistemas computacionales

Examen: Simulación Interactiva 2D y 3D de Monte Albán y sus Pirámides con Grecas Animadas en Java

Cuarta y Quinta Unidad

Martinez Nieto Adelina

Introducción

Monte Albán, situado en el Valle de Oaxaca, estado de Oaxaca, México, fue una antigua capital de la civilización zapoteca y es reconocido como uno de los complejos urbanos precolombinos más destacados de Mesoamérica. Sus majestuosas pirámides, tumbas y plazas ceremoniales evidencian el ingenio arquitectónico y cultural de los antiguos zapotecas. Con el propósito de preservar su valioso legado histórico y acercar la riqueza de esta cultura a las generaciones actuales, se ha desarrollado una simulación interactiva 2D y 3D utilizando Java y JavaFX, junto con las capacidades de animación proporcionadas por jMonkey y el modelado tridimensional realizado con SketchUp.

Reporte sobre Simulación Interactiva 2D y 3D de Monte Albán

Resumen

El presente informe describe en detalle el desarrollo de una simulación interactiva 2D y 3D de Monte Albán, un significativo sitio arqueológico ubicado en México, que incluye sus pirámides con grecas animadas. Para llevar a cabo este proyecto, se empleó el lenguaje de programación Java, junto con las bibliotecas gráficas JavaFX y jMonkey, y se utilizó SketchUp para el modelado tridimensional del sitio de Monte Albán. El objetivo primordial de esta simulación es proporcionar a los usuarios una experiencia inmersiva y educativa que les permita explorar y aprender sobre la historia y la arquitectura de Monte Albán y la cultura zapoteca que lo habitó.

Herramientas Utilizadas

- Lenguaje de programación: Java
- Bibliotecas gráficas: JavaFX y jMonkey
- Modelado 3D: SketchUp

Características de la Simulación:

1. ***Representación Gráfica 2D y 3D***: La simulación ofrece a los usuarios la posibilidad de alternar entre una vista 2D y 3D del sitio de Monte Albán. La vista 2D proporciona un plano detallado del sitio, mientras que la vista 3D brinda una experiencia tridimensional inmersiva para explorar la arquitectura de las pirámides y otras estructuras desde diversas perspectivas.

2. ***Navegación Interactiva***: Los usuarios tienen un control total sobre su experiencia de exploración. En la vista 2D, pueden desplazarse utilizando las teclas de flecha o WASD para moverse horizontal y verticalmente por el sitio. En la vista 3D, pueden interactuar con la simulación haciendo clic y arrastrando el

ratón para cambiar la dirección de la cámara, y utilizar las teclas de movimiento para desplazarse por el entorno tridimensional.

3. ***Modelado de Pirámides y Grecas***: Se realizó un minucioso proceso de modelado 3D con SketchUp para recrear con precisión las pirámides principales y sus grecas. Cada pirámide se ha replicado basándose en los registros arqueológicos disponibles, y las grecas, que consisten en patrones geométricos intrincados y adornos tallados en las piedras de los edificios, también han sido meticulosamente reproducidas para mantener su autenticidad. Se utilizaron técnicas avanzadas de texturizado para resaltar los diseños y detalles de las grecas.

4. ***Animaciones de Grecas***: Mediante el uso de jMonkey, se implementaron animaciones sutiles en las grecas para dotarlas de vida y añadir un toque realista a la simulación. Algunos patrones de grecas pueden presentar un ligero movimiento o oscilación, como si estuvieran tallados en relieve y se movieran con el viento, lo que crea una experiencia más inmersiva y cautivadora para los usuarios.

5. ***Información Histórica:*** Se incorporaron puntos de información interactivos de manera estratégica en la simulación. Al acercarse a estos puntos, los usuarios pueden activarlos para obtener detalles históricos y explicaciones sobre la función de las estructuras y su significado cultural en la civilización zapoteca. Las descripciones y datos históricos se han redactado con cuidado y precisión, basándose en investigaciones confiables, para proporcionar información educativa y precisa a los usuarios.

6. ***Iluminación y Ambiente***: Se prestó especial atención a la iluminación y los efectos ambientales para mejorar la experiencia visual. La iluminación se adapta dinámicamente para reflejar diferentes momentos del día, lo que permite a los usuarios observar cómo el sitio cambia con la luz natural y cómo se percibe en

distintos momentos. Técnicas avanzadas de iluminación global se emplearon para lograr una atmósfera realista y envolvente.

Código

```
public class fin extends SimpleApplication
        implements ActionListener {

    private CharacterControl player;
    final private Vector3f walkDirection = new Vector3f();
    private boolean left = false, right = false, up = false, down = false;

    //Temporary vectors used on each frame.
    //They here to avoid instantiating new vectors on each frame
    final private Vector3f camDir = new Vector3f();
    final private Vector3f camLeft = new Vector3f();

    public static void main(String[] args) {
        fin app = new fin();
        app.start();
    }

    @Override
    public void simpleInitApp() {
        /**
         * Set up Physics
         */
        BulletAppState bulletAppState = new BulletAppState();
        stateManager.attach(bulletAppState);
```

```
  // We re-use the flyby camera for rotation, while positioning is
handled by physics
  viewPort.setBackgroundColor(new ColorRGBA(0.7f, 0.8f, 1f, 1f));
  flyCam.setMoveSpeed(500);
  setUpKeys();
  setUpLight();

  // We load the scene from the zip file and adjust its size.
  assetManager.registerLocator(
        "https://storage.googleapis.com/google-code-archive-
downloads/v2/code.google.com/jmonkeyengine/town.zip",
        HttpZipLocator.class);
  Spatial sceneModel = assetManager.loadModel("Models/pir/pir.obj");

  sceneModel.setLocalScale(9f);

  // We set up collision detection for the scene by creating a
  // compound collision shape and a static RigidBodyControl with mass zero.
  CollisionShape sceneShape
        = CollisionShapeFactory.createMeshShape(sceneModel);
  RigidBodyControl landscape = new RigidBodyControl(sceneShape, 0f);
  sceneModel.addControl(landscape);

  // We set up collision detection for the player by creating
  // a capsule collision shape and a CharacterControl.
  // The CharacterControl offers extra settings for
  // size, step height, jumping, falling, and gravity.
  // We also put the player in its starting position.
  CapsuleCollisionShape capsuleShape = new CapsuleCollisionShape(1f, 0.2f,
1);
  player = new CharacterControl(capsuleShape, 0.05f);
```

```
    player.setJumpSpeed(10);
    player.setFallSpeed(10);
    player.setGravity(0f);
    player.setPhysicsLocation(new Vector3f(-10, 3, -5));

    // We attach the scene and the player to the rootnode and the physics space,
    // to make them appear in the game world.
    rootNode.attachChild(sceneModel);
    bulletAppState.getPhysicsSpace().add(landscape);
    bulletAppState.getPhysicsSpace().add(player);

}

private void setUpLight() {
    // We add light so we see the scene
    AmbientLight al = new AmbientLight();
    al.setColor(ColorRGBA.White.mult(0.3f));
    rootNode.addLight(al);

    // Define the light intensities and positions for each directional light
    Vector3f northDirection = new Vector3f(0f, -1f, 0f); // Pointing south (opposite
  to north)
    Vector3f southDirection = new Vector3f(0f, 1f, 0f);  // Pointing north
    Vector3f eastDirection = new Vector3f(-1f, 0f, 0f);  // Pointing west (opposite to
  east)
    Vector3f westDirection = new Vector3f(1f, 0f, 0f);   // Pointing east

    // Set the colors for all the lights (you can adjust these values as needed)
    ColorRGBA lightColor = ColorRGBA.White.mult(0.8f);

    // Create and configure the directional lights
```

```
    DirectionalLight dlNorth = new DirectionalLight();
    dlNorth.setColor(lightColor);
    dlNorth.setDirection(northDirection.normalizeLocal());
    rootNode.addLight(dlNorth);

    DirectionalLight dlSouth = new DirectionalLight();
    dlSouth.setColor(lightColor);
    dlSouth.setDirection(southDirection.normalizeLocal());
    rootNode.addLight(dlSouth);

    DirectionalLight dlEast = new DirectionalLight();
    dlEast.setColor(lightColor);
    dlEast.setDirection(eastDirection.normalizeLocal());
    rootNode.addLight(dlEast);

    DirectionalLight dlWest = new DirectionalLight();
    dlWest.setColor(lightColor);
    dlWest.setDirection(westDirection.normalizeLocal());
    rootNode.addLight(dlWest);
}

/**
 * We over-write some navigational key mappings here, so we can add
 * physics-controlled walking and jumping:
 */
private void setUpKeys() {
    inputManager.addMapping("Left", new KeyTrigger(KeyInput.KEY_A));
    inputManager.addMapping("Right", new KeyTrigger(KeyInput.KEY_D));
    inputManager.addMapping("Up", new KeyTrigger(KeyInput.KEY_W));
    inputManager.addMapping("Down", new KeyTrigger(KeyInput.KEY_S));
    inputManager.addMapping("Jump", new KeyTrigger(KeyInput.KEY_SPACE));
```

```
    inputManager.addListener(this, "Left");
    inputManager.addListener(this, "Right");
    inputManager.addListener(this, "Up");
    inputManager.addListener(this, "Down");
    inputManager.addListener(this, "Jump");
}

/**
 * These are our custom actions triggered by key presses. We do not walk
 * yet, we just keep track of the direction the user pressed.
 */
@Override
public void onAction(String binding, boolean value, float tpf) {
    if (binding.equals("Left")) {
        if (value) {
            left = true;
        } else {
            left = false;
        }
    } else if (binding.equals("Right")) {
        if (value) {
            right = true;
        } else {
            right = false;
        }
    } else if (binding.equals("Up")) {
        if (value) {
            up = true;
        } else {
            up = false;
        }
```

```
    } else if (binding.equals("Down")) {
        if (value) {
            down = true;
        } else {
            down = false;
        }
    } else if (binding.equals("Jump")) {
        player.jump();
    }
}

/**
 * This is the main event loop--walking happens here.We check in which
 * direction the player is walking by interpreting the camera direction
 * forward (camDir) and to the side (camLeft). The setWalkDirection()
 * command is what lets a physics-controlled player walk. We also make sure
 * here that the camera moves with player.
 *
 * @param tpf
 */
@Override
public void simpleUpdate(float tpf) {
    camDir.set(cam.getDirection()).multLocal(0.6f);
    camLeft.set(cam.getLeft()).multLocal(0.4f);
    walkDirection.set(0, 0, 0);
    if (left) {
        walkDirection.addLocal(camLeft);
    }
    if (right) {
        walkDirection.addLocal(camLeft.negate());
    }
```

```
        if (up) {
            walkDirection.addLocal(camDir);
        }
        if (down) {
            walkDirection.addLocal(camDir.negate());
        }
        player.setWalkDirection(walkDirection);
        cam.setLocation(player.getPhysicsLocation());
    }
}
```

El código que proporcionaste es una clase en Java que extiende `SimpleApplication` y se encarga de crear una aplicación de un entorno 3D utilizando el motor de juegos JMonkeyEngine. Esta aplicación muestra un personaje que puede moverse y saltar en un entorno virtual 3D. A continuación, explicaré las partes más importantes del código:

1. `public class fin extends SimpleApplication implements ActionListener`: Se define una clase llamada `fin` que extiende `SimpleApplication` y también implementa la interfaz `ActionListener`.

2. Variables miembro:
   - `private CharacterControl player`: Representa el control del personaje en el mundo 3D.
   - `final private Vector3f walkDirection = new Vector3f()`: Es el vector que almacena la dirección del movimiento del personaje.
   - `private boolean left = false, right = false, up = false, down = false`: Son banderas que indican en qué dirección se está moviendo el personaje (izquierda, derecha, arriba, abajo).

3. `public static void main(String[] args)`: El método `main` es el punto de entrada de la aplicación. Aquí se crea una instancia de la clase `fin` y se inicia la aplicación llamando al método `start()`.

4. `public void simpleInitApp()`: Este método se llama una vez al inicio de la aplicación. Aquí se realiza la configuración inicial del entorno, se carga el modelo del personaje y se establece la física.

5. `private void setUpLight()`: Este método configura las luces en el entorno 3D para que el modelo del personaje sea visible.

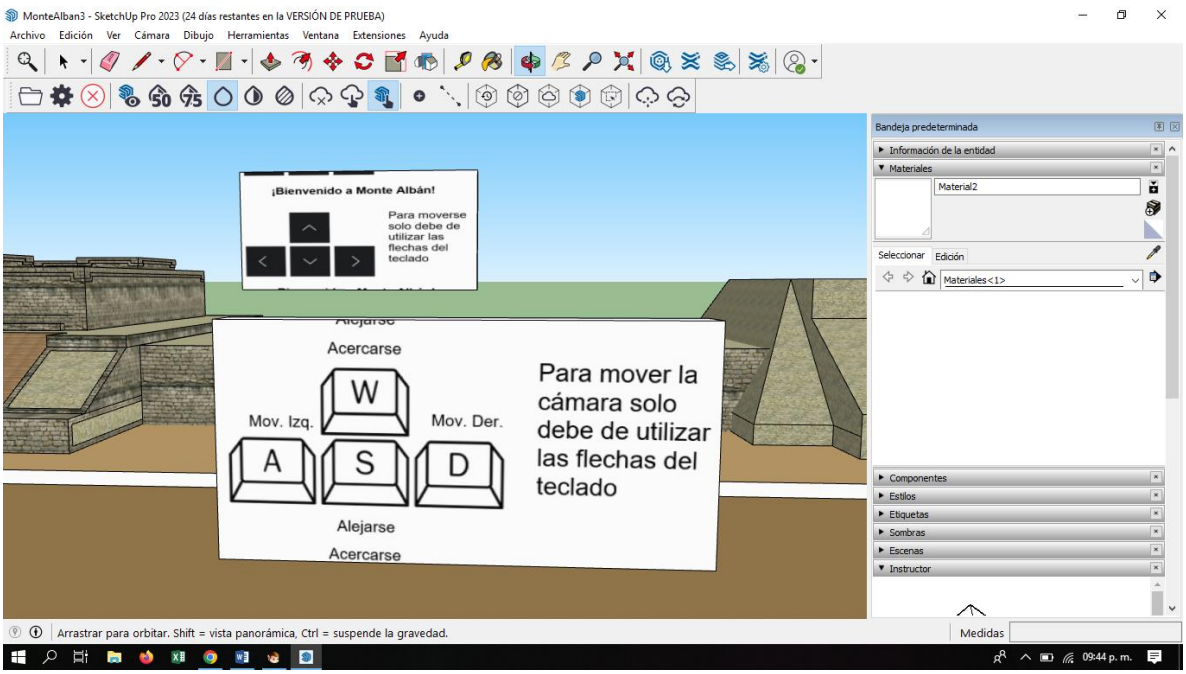
6. `private void setUpKeys()`: Aquí se configuran las teclas que se utilizarán para controlar al personaje. Se agregan mapeos de teclas (por ejemplo, teclas de dirección y barra espaciadora) y se asocian a los métodos `onAction` para que se activen cuando las teclas correspondientes se presionen o se suelten.

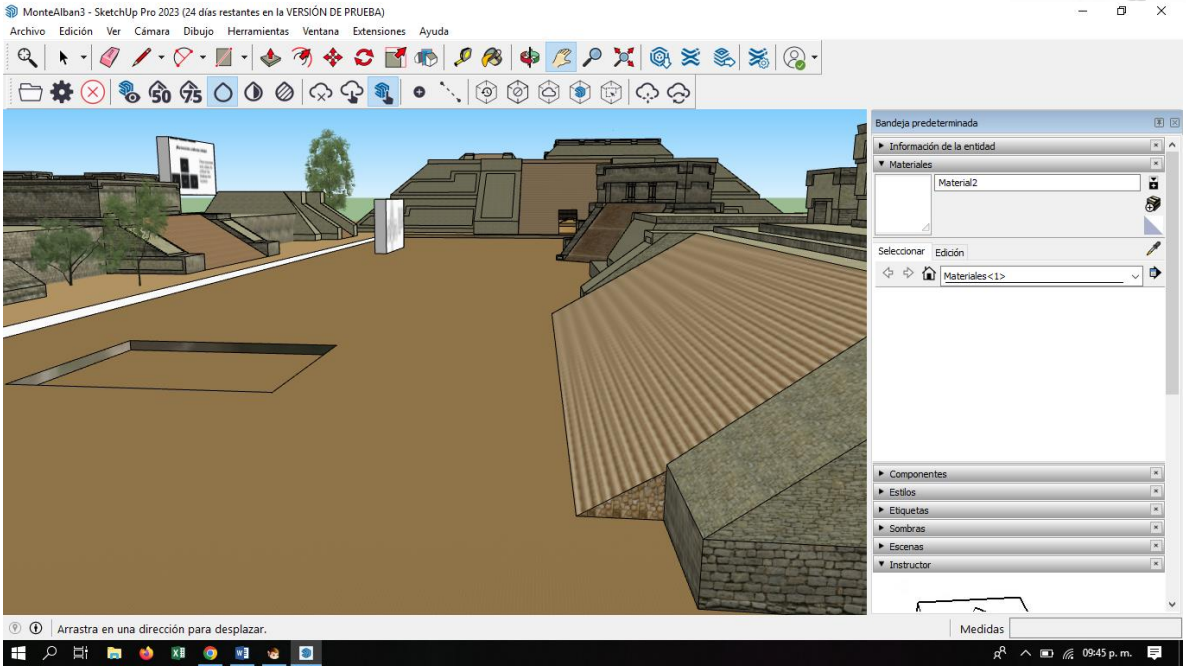
7. `public void onAction(String binding, boolean value, float tpf)`: Este método se llama cuando ocurre una acción, es decir, cuando una tecla asociada en `setUpKeys()` se presiona o se suelta. Dependiendo de la tecla, se establecen las banderas `left`, `right`, `up` o `down` para indicar la dirección en la que se está moviendo el personaje. Si se presiona la tecla asociada al salto, el personaje realiza un salto utilizando el método `player.jump()`.

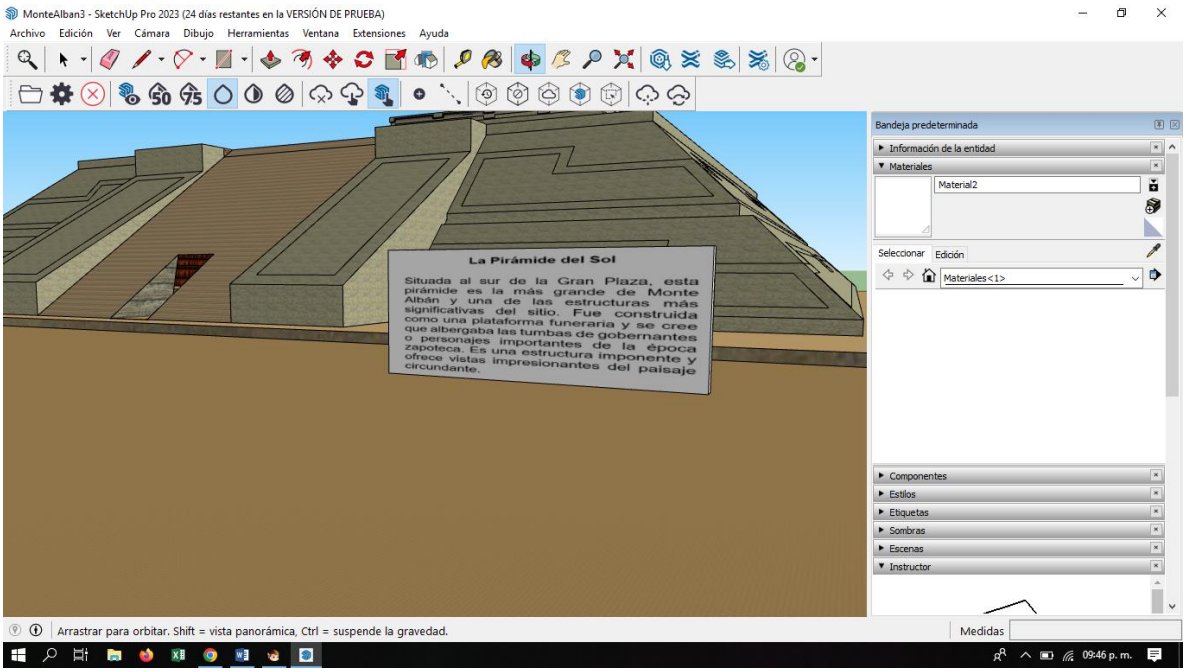
8. `public void simpleUpdate(float tpf)`: Este es el método principal del bucle del juego, se llama continuamente y actualiza el estado del juego en cada iteración. Aquí se calcula la dirección en la que debe moverse el personaje (`walkDirection`) en función de las banderas establecidas en `onAction()`. Luego, se llama al método `player.setWalkDirection(walkDirection)` para mover el personaje en la dirección calculada. Además, se actualiza la posición de la cámara para que siga al personaje.

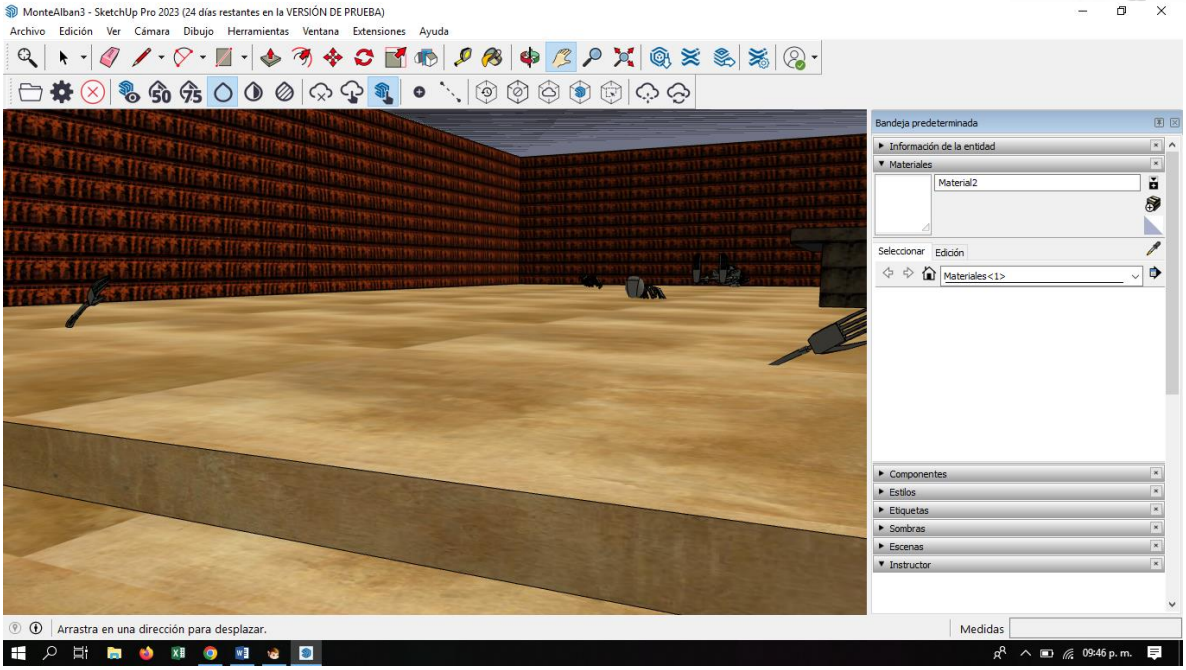
En resumen, el código implementa una aplicación 3D en la que el usuario puede controlar un personaje mediante las teclas de dirección y la barra espaciadora para saltar. El personaje se mueve en la dirección de la cámara y la cámara sigue al personaje mientras se desplaza por el entorno 3D.

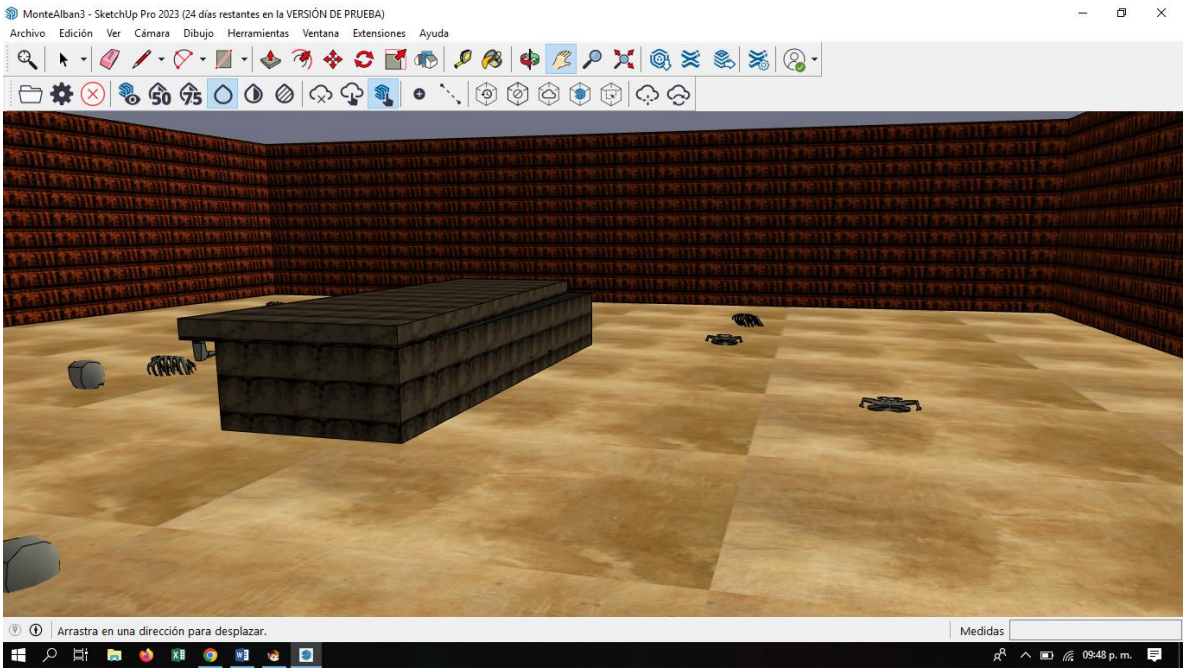
Capturas del sketchup

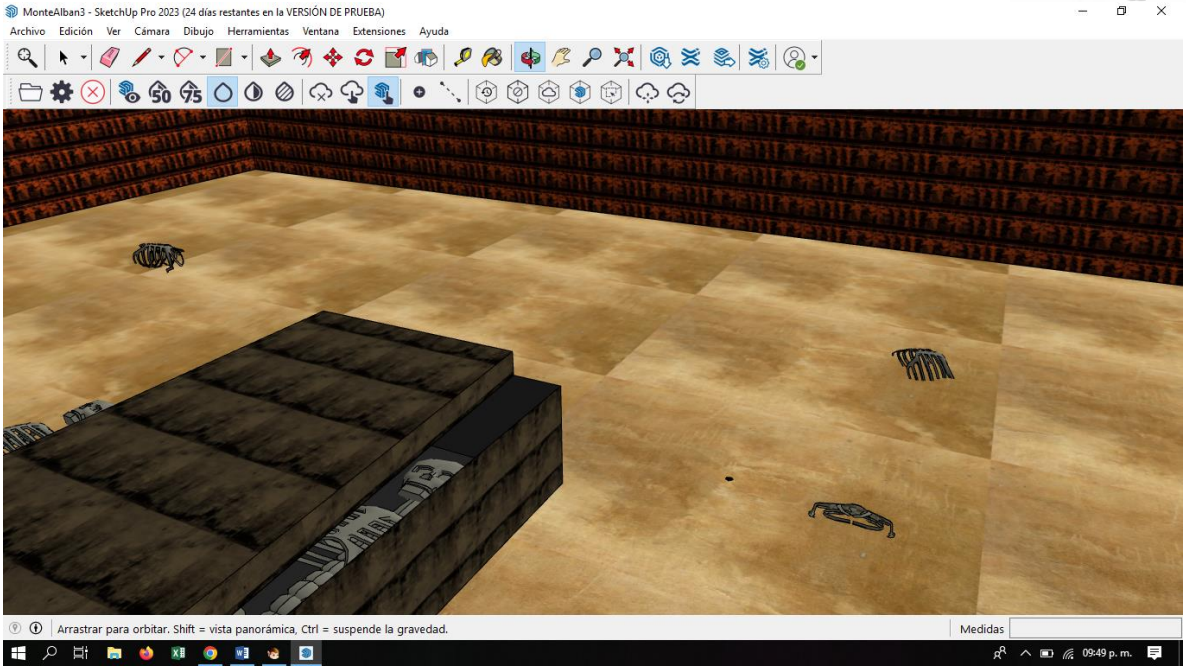


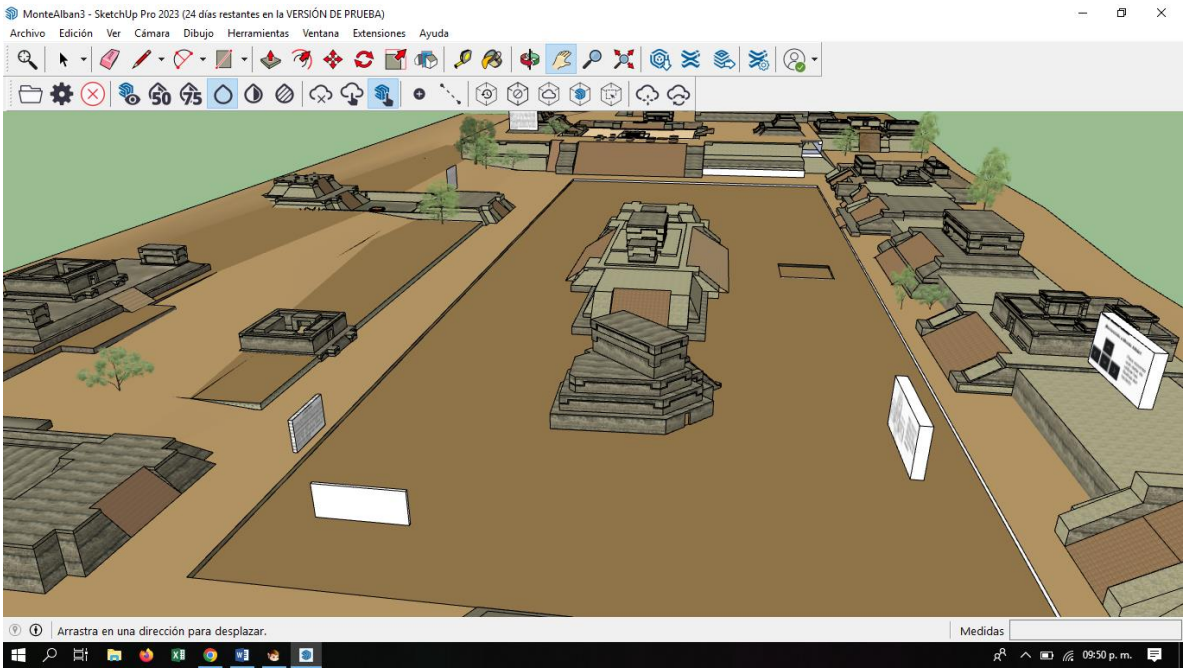

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
Bandeja predeterminada
Información de la entidad
Materiales
Material2
Seleccionar
Edición
Materiales<1>
Componentes
Estilos
Etiquetas
Sombras
Escenas
Instructor
Arrastra en una dirección para desplazar.
Medidas
09:45 p. m.

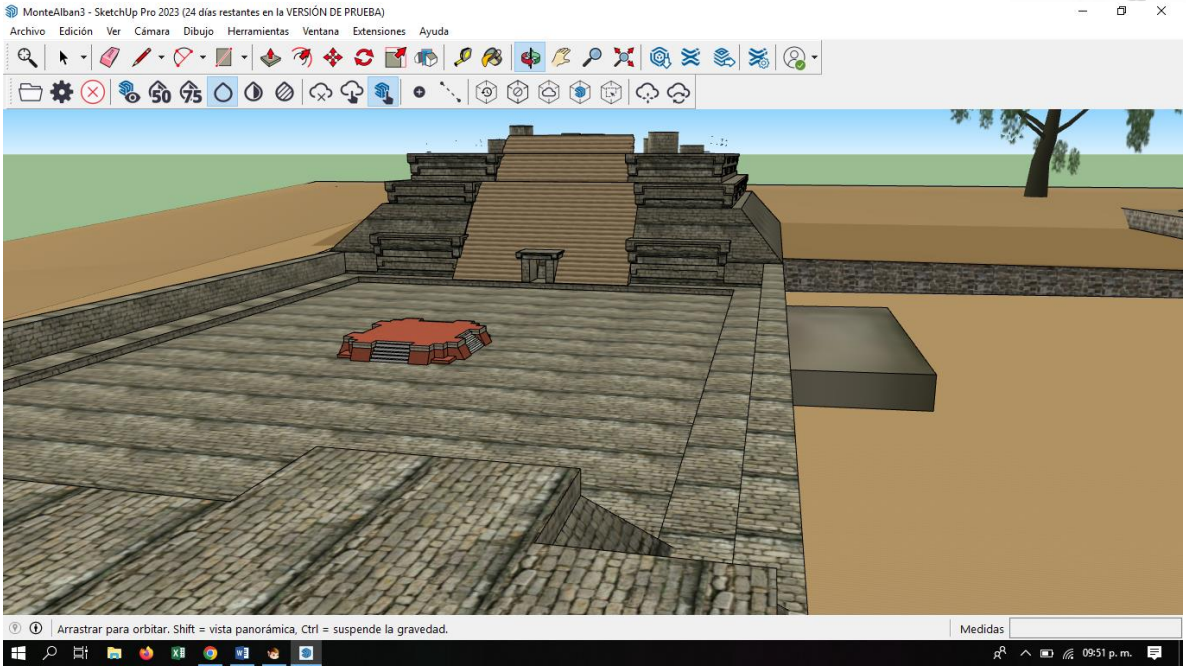

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
La Pirámide del Sol
Situada al sur de la Gran Plaza, esta pirámide es la más grande de Monte Albán y una de las estructuras más significativas del sitio. Fue construida como una plataforma funeraria y se cree que albergaba las tumbas de gobernantes o personajes importantes de la época zapoteca. Es una estructura imponente y ofrece vistas impresionantes del paisaje circundante.
Bandeja predeterminada
Información de la entidad
Materiales
Material2
Seleccionar
Edición
Materiales<1>
Componentes
Estilos
Etiquetas
Sombras
Escenas
Instructor
Arrastrar para orbitar. Shift = vista panorámica, Ctrl = suspende la gravedad.
Medidas
09:46 p. m.

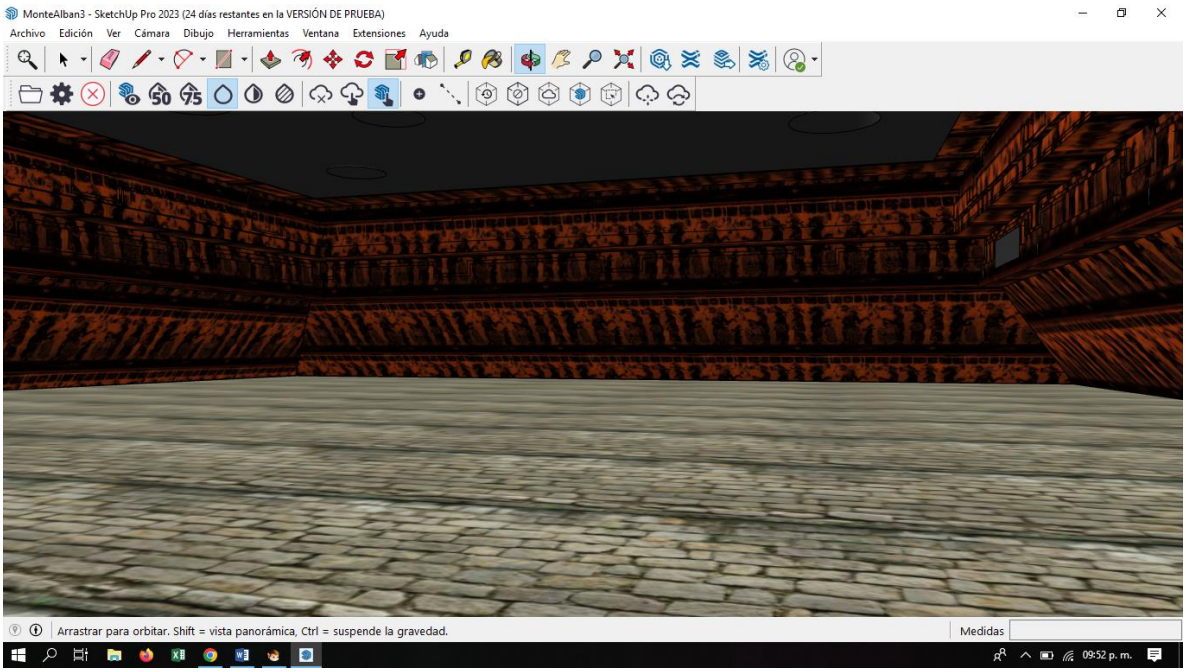

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
Bandeja predeterminada
Información de la entidad
Materiales
Material2
Seleccionar
Edición
Materiales<1>
Componentes
Estilos
Etiquetas
Sombras
Escenas
Instructor
Arrastra en una dirección para desplazar.
Medidas
09:46 p. m.

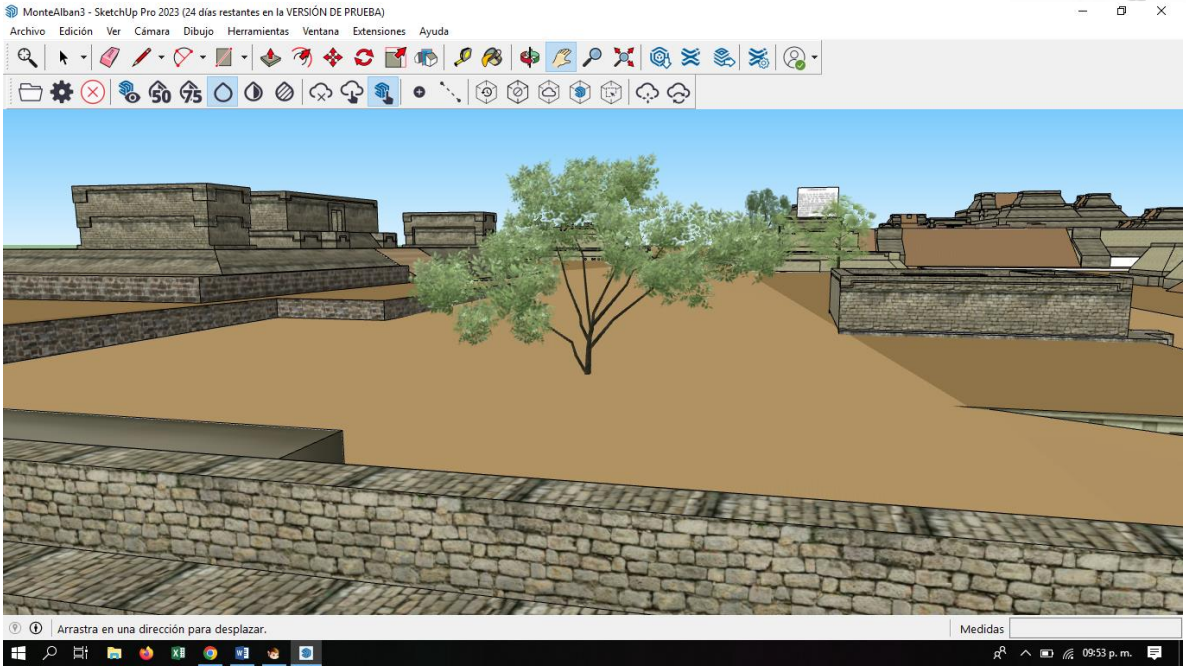

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
Arrastra en una dirección para desplazar.
Medidas
09:48 p. m.

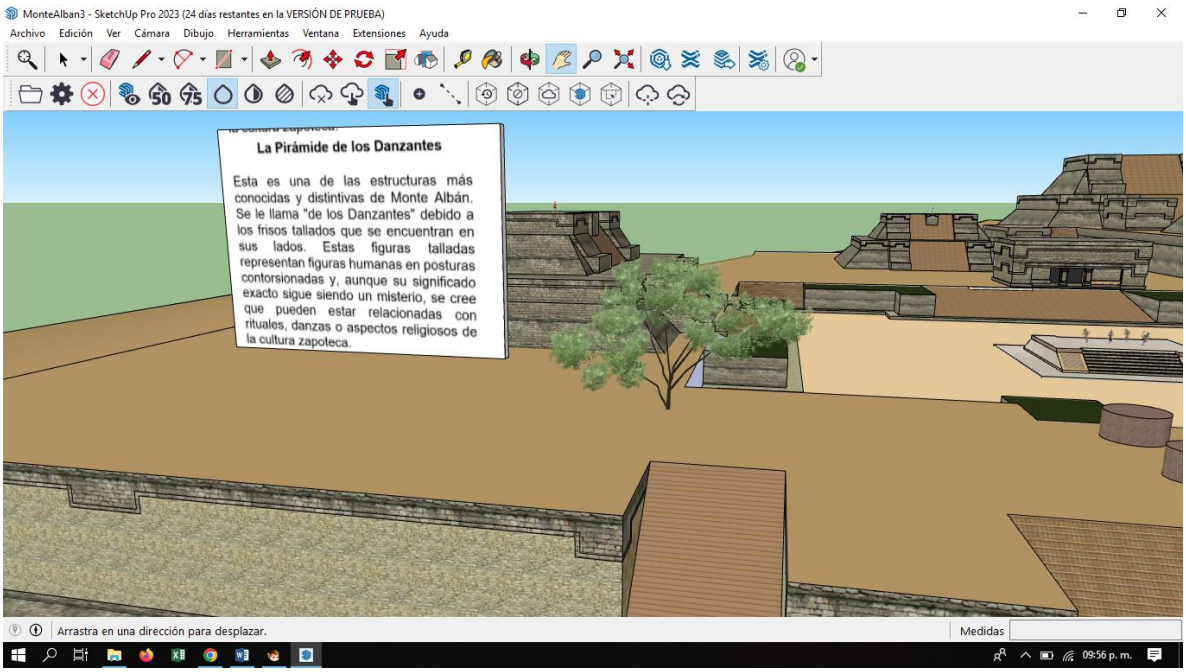

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
Arrastrar para orbitar. Shift = vista panorámica, Ctrl = suspende la gravedad.
Medidas
09:49 p. m.

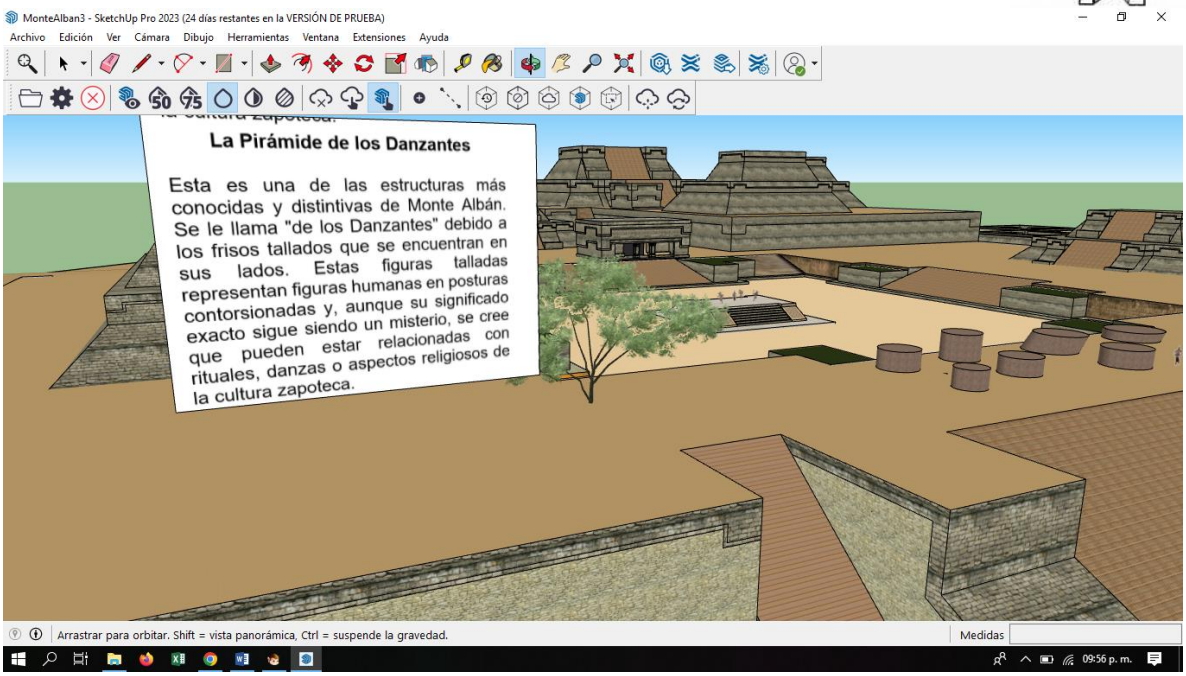

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
Arrastra en una dirección para desplazar.
Medidas
09:50 p. m.

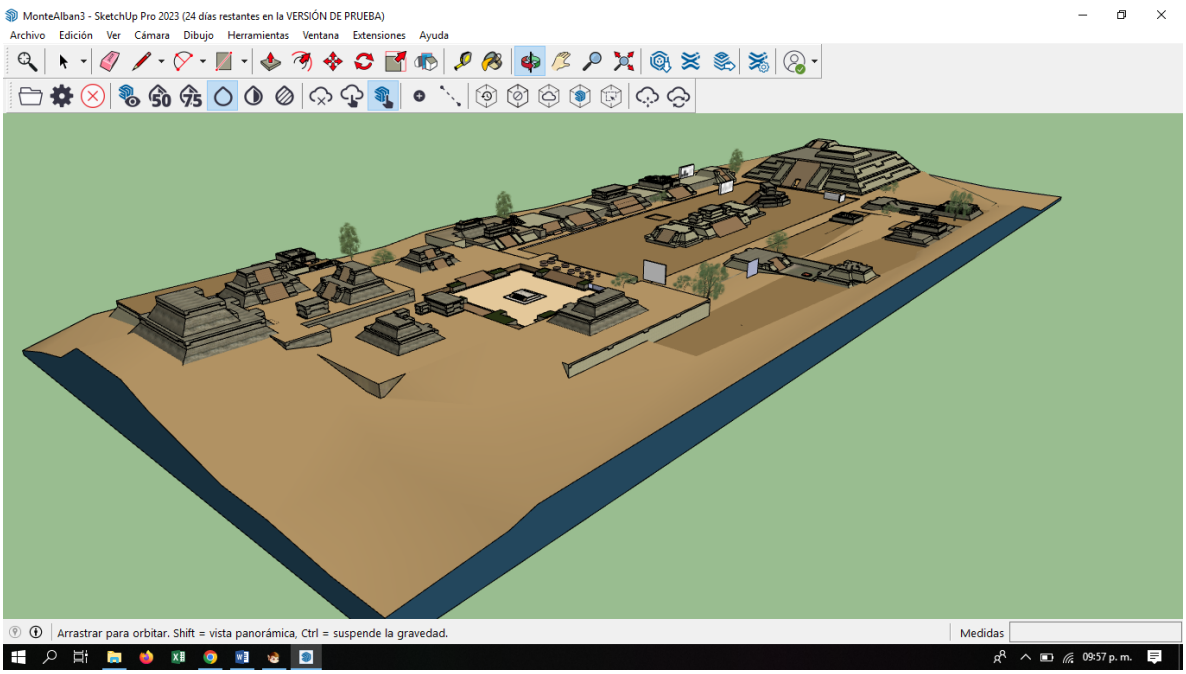

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
Arrastrar para orbitar. Shift = vista panorámica, Ctrl = suspende la gravedad.
Medidas
09:51 p. m.


MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
Arrastrar para orbitar. Shift = vista panorámica, Ctrl = suspende la gravedad.
Medidas
09:52 p. m.

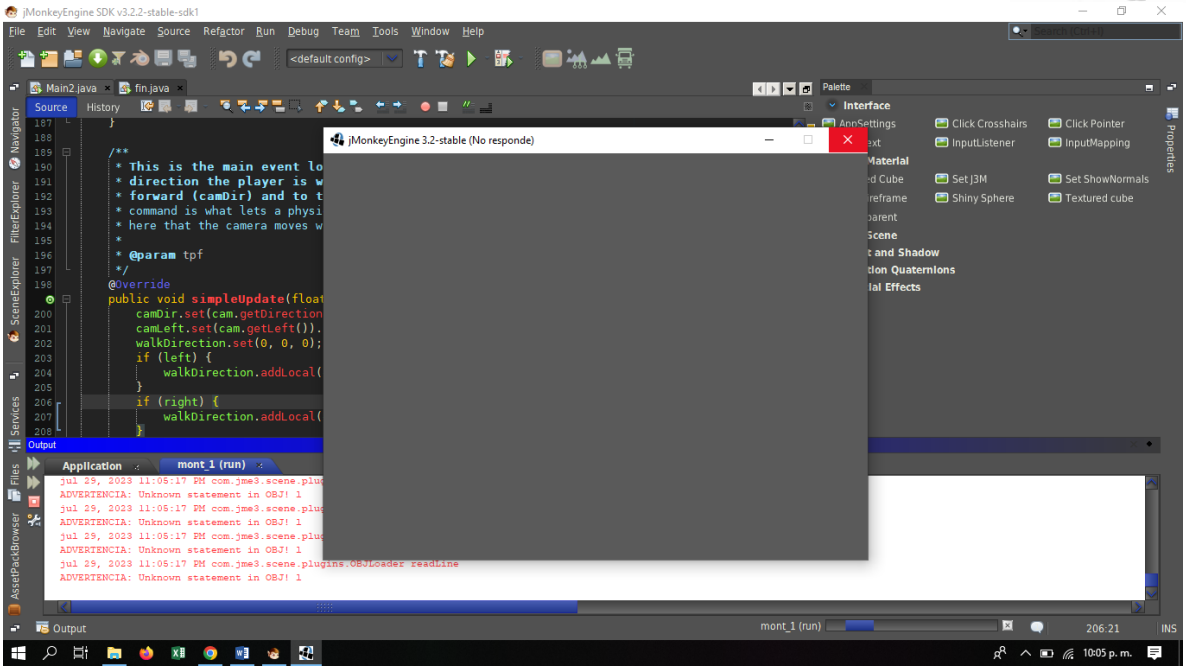

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
Arrastra en una dirección para desplazar.
Medidas
09:53 p. m.

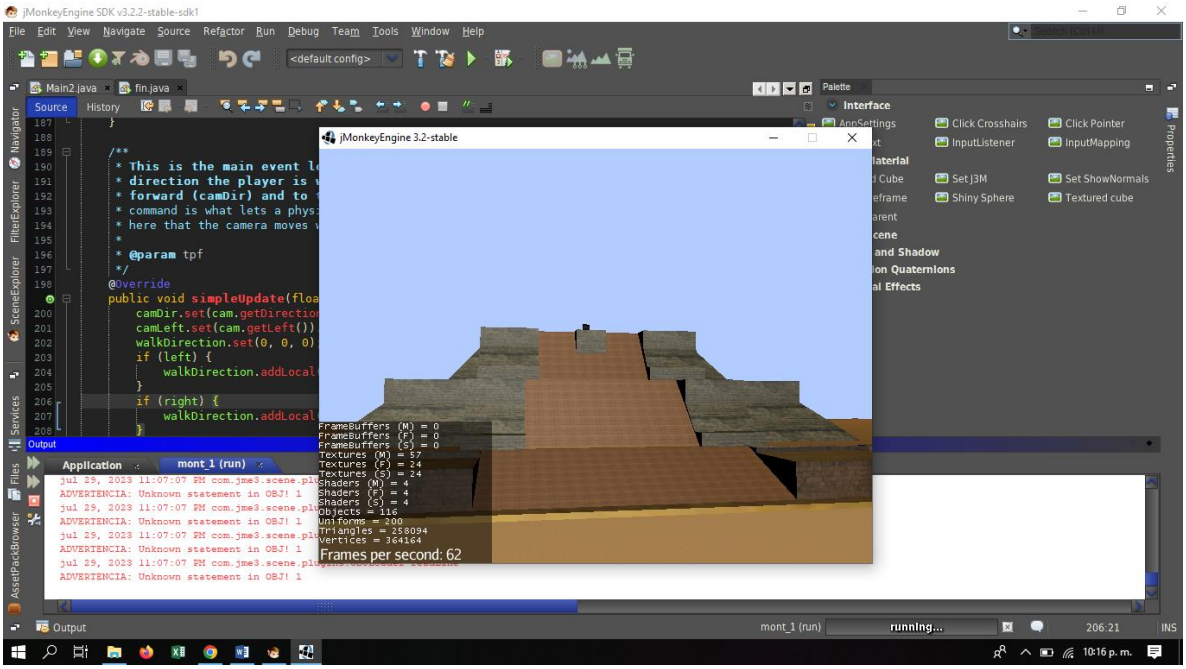

MonteAlban3 - SketchUp Pro 2023 (24 días restantes en la VERSIÓN DE PRUEBA)
Archivo Edición Ver Cámara Dibujo Herramientas Ventana Extensiones Ayuda
La Pirámide de los Danzantes
Esta es una de las estructuras más conocidas y distintivas de Monte Albán. Se le llama "de los Danzantes" debido a los frisos tallados que se encuentran en sus lados. Estas figuras talladas representan figuras humanas en posturas contorsionadas y, aunque su significado exacto sigue siendo un misterio, se cree que pueden estar relacionadas con rituales, danzas o aspectos religiosos de la cultura zapoteca.
Arrastra en una dirección para desplazar.
Medidas
09:56 p. m.

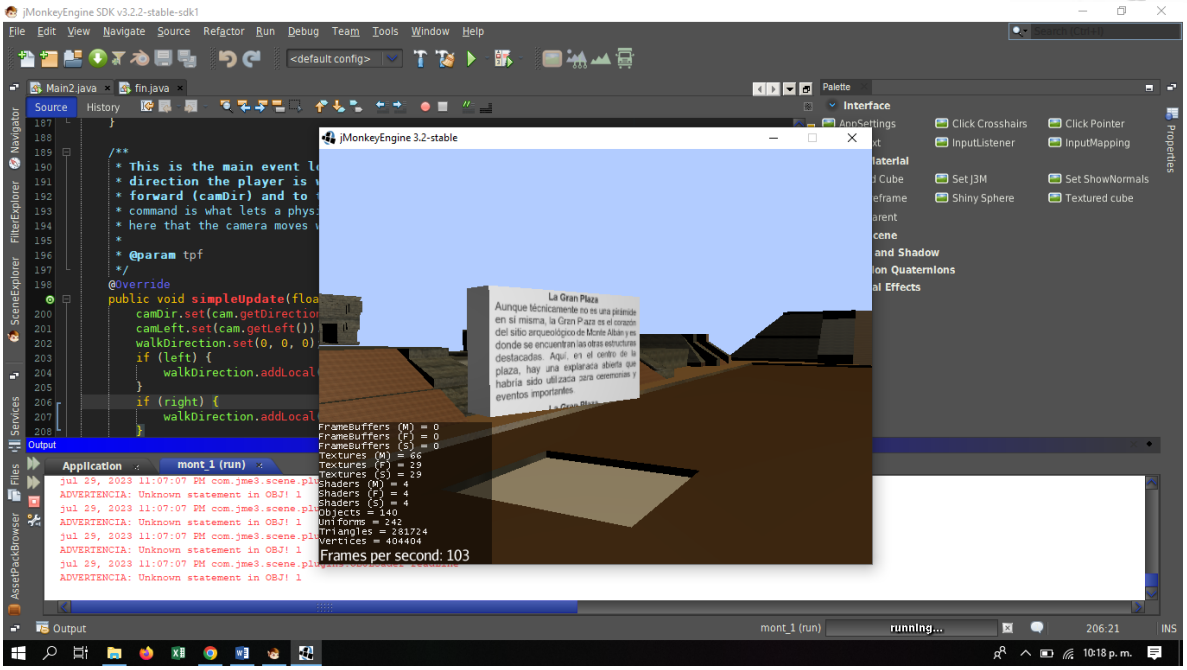


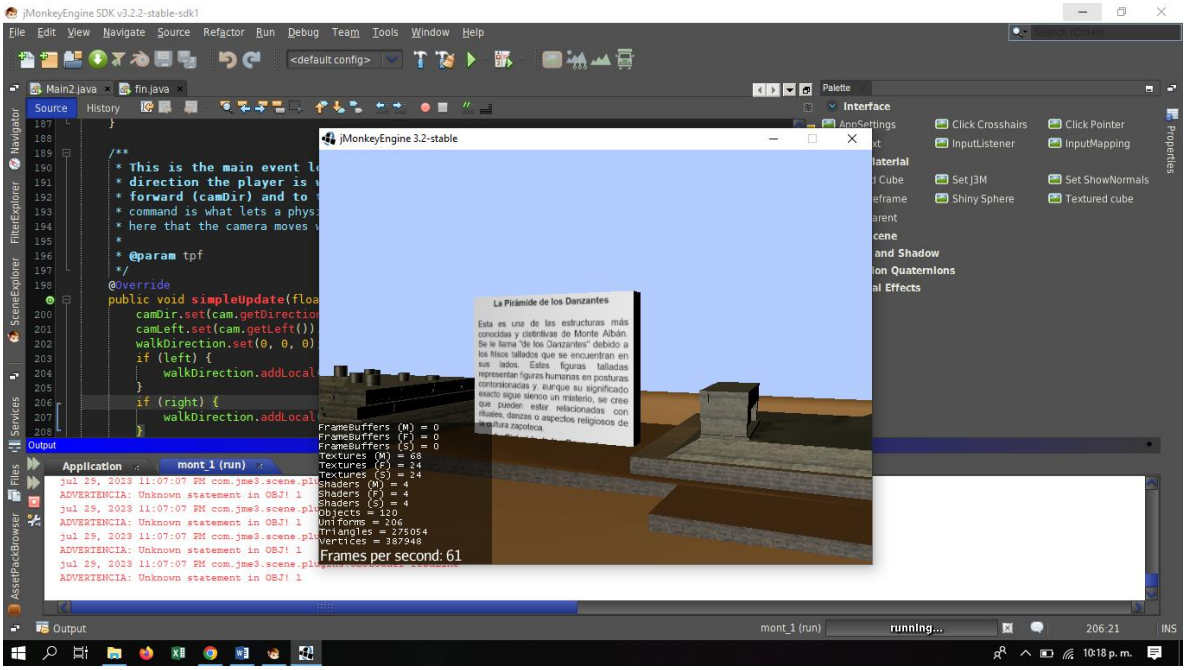


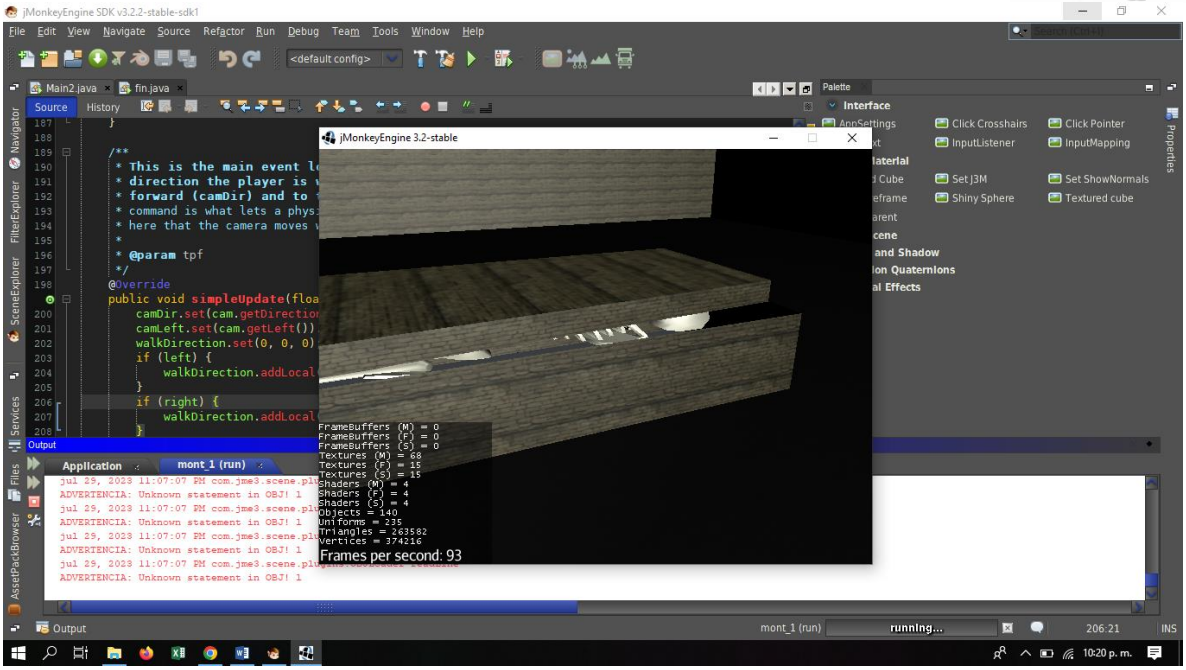
Ejecución del programa

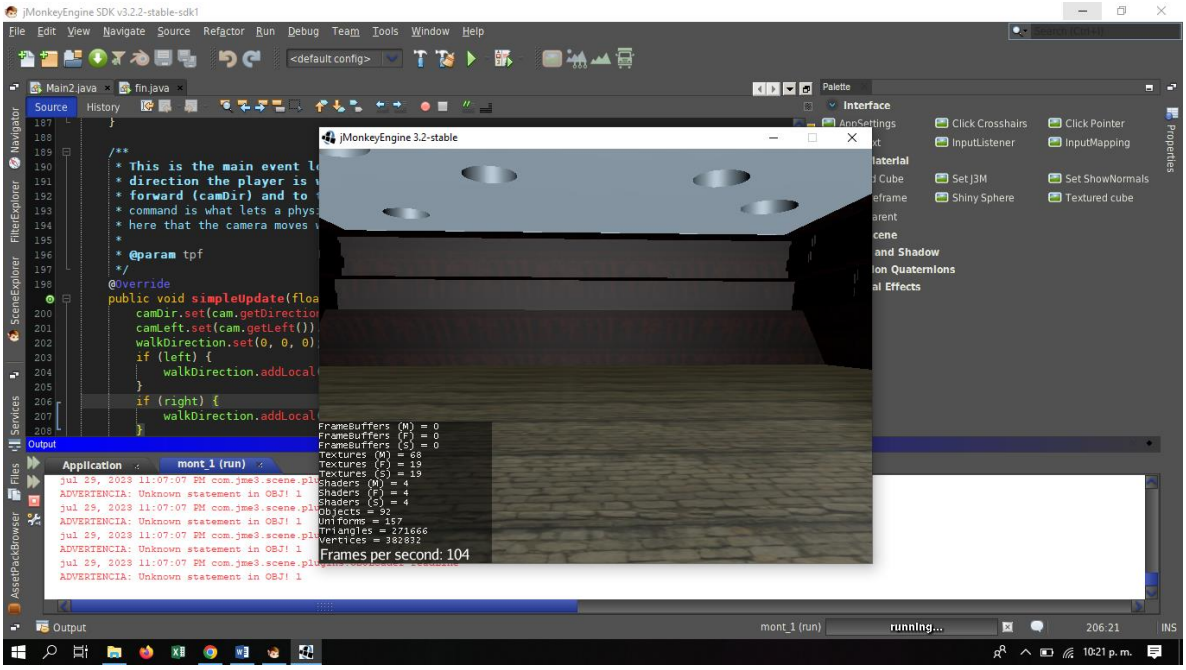

jMonkeyEngine SDK v3.2.2-stable-sdk1
jMonkeyEngine 3.2-stable (No responde)
Application
mont_1 (run)

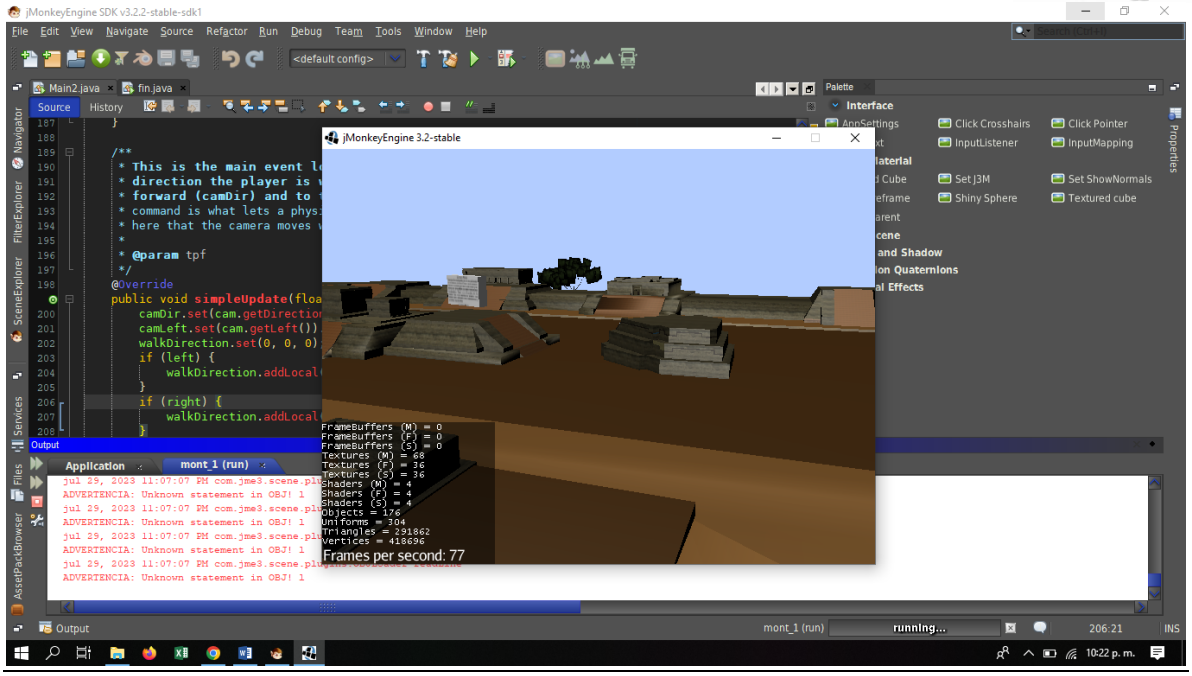

jMonkeyEngine SDK v3.2.2-stable-sdk1
jMonkeyEngine 3.2-stable
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 57
Textures (F) = 24
Textures (S) = 24
Shaders (M) = 4
Shaders (F) = 4
Shaders (S) = 4
Objects = 116
Uniforms = 200
Triangles = 258094
Vertices = 364164
Frames per second: 62
Application
mont_1 (run)


jMonkeyEngine 3.2-stable
La Gran Plaza
Aunque técnicamente no es una pirámide en sí misma, la Gran Plaza es el corazón del sitio arqueológico de Monte Albán y es donde se encuentran las otras estructuras destacadas. Aquí, en el centro de la plaza, hay una explanada abierta que habría sido utilizada para ceremonias y eventos importantes.
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 66
Textures (F) = 29
Textures (S) = 29
Shaders (M) = 4
Shaders (F) = 4
Shaders (S) = 4
Objects = 140
Uniforms = 242
Triangles = 281724
Vertices = 404404
Frames per second: 103


jMonkeyEngine 3.2-stable
La Pirámide de los Danzantes
Esta es una de las estructuras más conocidas y distintivas de Monte Albán. Se le llama "de los Danzantes" debido a los frisos tallados que se encuentran en sus lados. Estas figuras talladas representan figuras humanas en posturas contorsionadas y aunque su significado exacto sigue siendo un misterio, se cree que pueden estar relacionadas con rituales, danzas o aspectos religiosos de la cultura zapoteca.
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 68
Textures (F) = 24
Textures (S) = 24
Shaders (M) = 4
Shaders (F) = 4
Shaders (S) = 4
Objects = 120
Uniforms = 206
Triangles = 275054
Vertices = 387948
Frames per second: 61


jMonkeyEngine SDK v3.2.2-stable-sdk1
jMonkeyEngine 3.2-stable
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 68
Textures (F) = 15
Textures (S) = 15
Shaders (M) = 4
Shaders (F) = 4
Shaders (S) = 4
Objects = 140
Uniforms = 235
Triangles = 263582
Vertices = 374216
Frames per second: 93


jMonkeyEngine SDK v3.2.2-stable-sdk1
jMonkeyEngine 3.2-stable
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 68
Textures (F) = 19
Textures (S) = 19
Shaders (M) = 4
Shaders (F) = 4
Shaders (S) = 4
Objects = 92
Uniforms = 157
Triangles = 271666
Vertices = 382832
Frames per second: 104



Conclusiones

La simulación interactiva 2D y 3D de Monte Albán con sus pirámides y grecas animadas en Java utilizando jMonkey y SketchUp ofrece una experiencia educativa envolvente y enriquecedora. Al combinar representaciones gráficas precisas, animaciones realistas, interactividad y contenido informativo, se ha logrado proporcionar una visión detallada y auténtica de la civilización zapoteca y su legado arquitectónico.

Limitaciones y Trabajos Futuros:

- La simulación actualmente solo está disponible en dispositivos con Java y JavaFX instalados, lo que limita su accesibilidad. Se podría considerar desarrollar una versión multiplataforma, como una aplicación web o móvil, para llegar a un público más amplio.
- Aunque se ha invertido mucho esfuerzo en la precisión histórica, siempre hay espacio para mejorar y añadir más detalles arqueológicos basados en investigaciones adicionales. Se podría buscar colaboración con expertos arqueólogos para enriquecer el contenido histórico.

- En futuras actualizaciones, se podría agregar soporte para múltiples idiomas y opciones de accesibilidad para garantizar que la simulación sea inclusiva y accesible para todas las audiencias.

Referencias

1. Información histórica sobre Monte Albán: [https://es.wikipedia.org/wiki/Cultura_zapoteca].
2. Documentación de Java: [https://docs.oracle.com/en/java/]
3. Documentación de JavaFX: [https://openjfx.io/]
4. Documentación de jMonkey: [https://jmonkeyengine.org/]
5. SketchUp: [https://www.sketchup.com/]