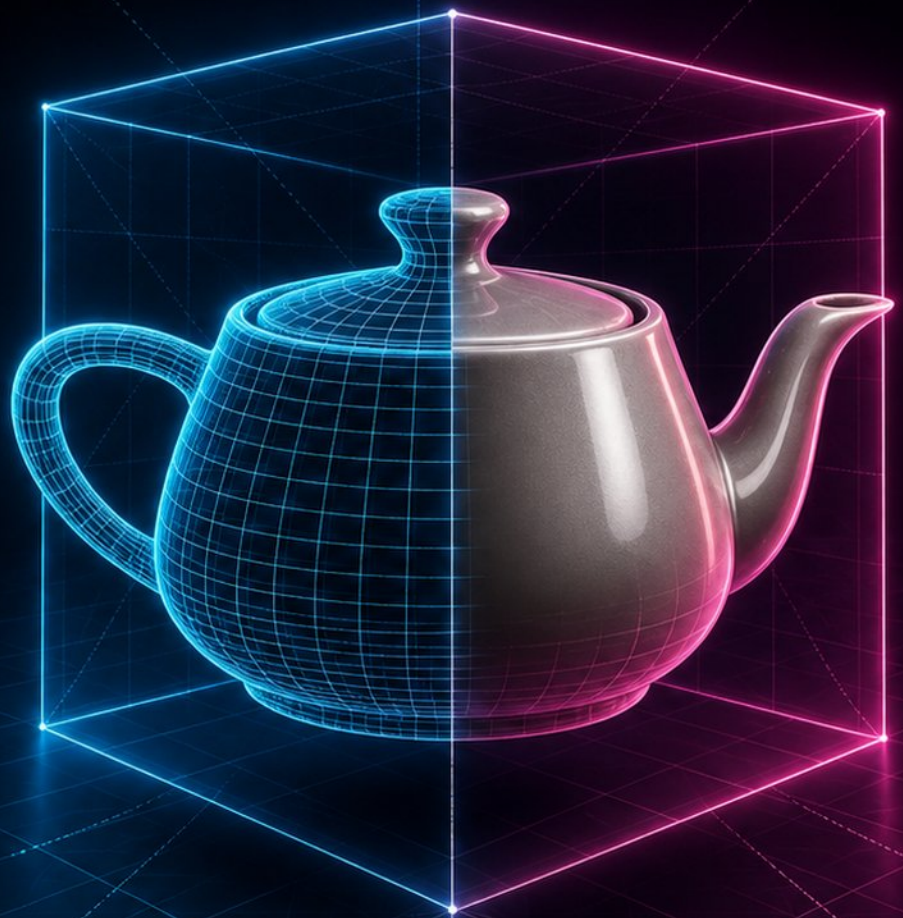
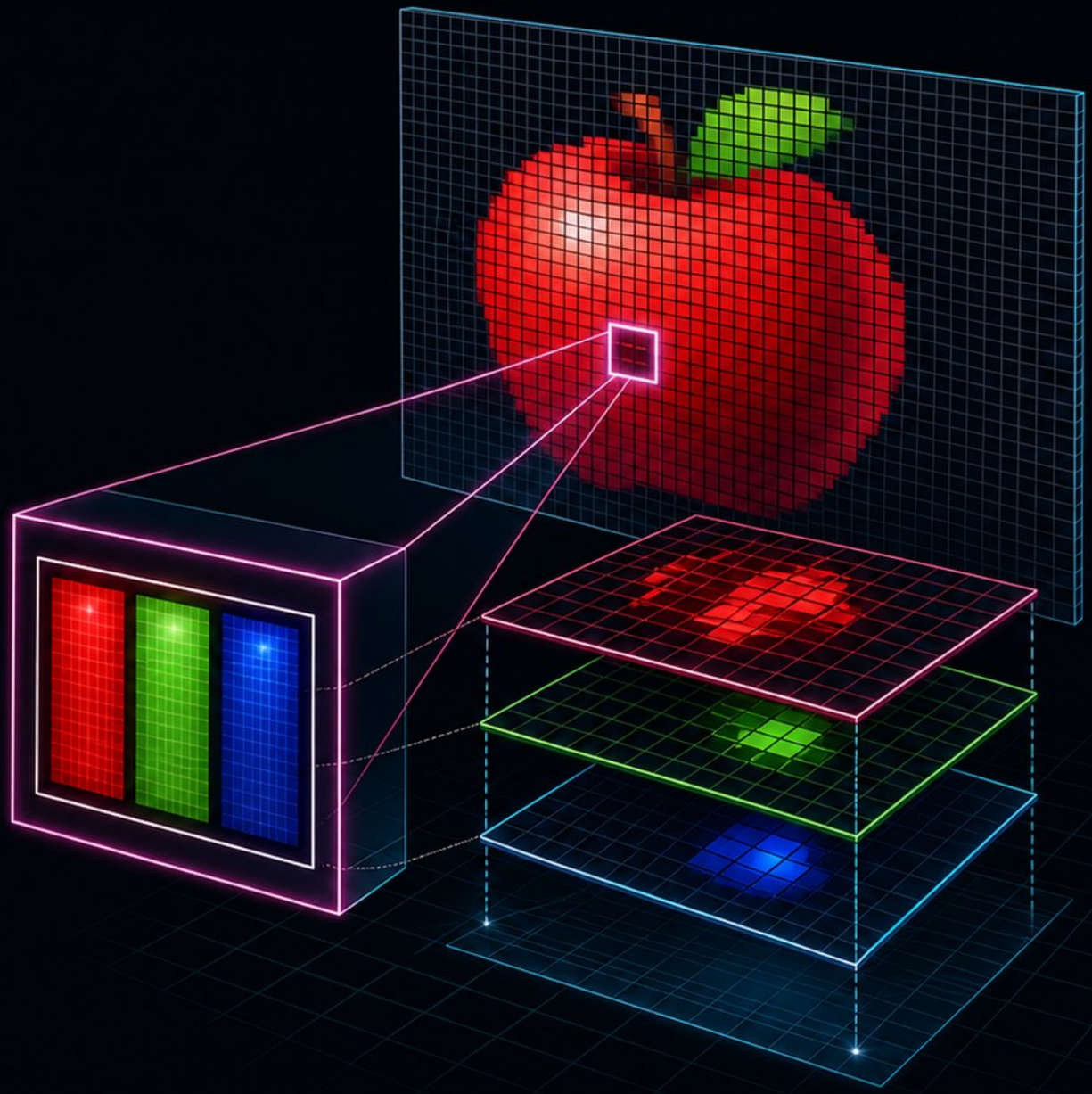
Introduction to Computer Graphics

- Computer graphics: creating digital images from 3D scenes, transforms, and rendering
- Understanding the pipeline bridges seeing an image and knowing how it was built
- Everyday examples: animated films, games, product visualizations, AR/VR, camera effects
- Roadmap: scene description → transforms → cameras → rendering → real-world applications



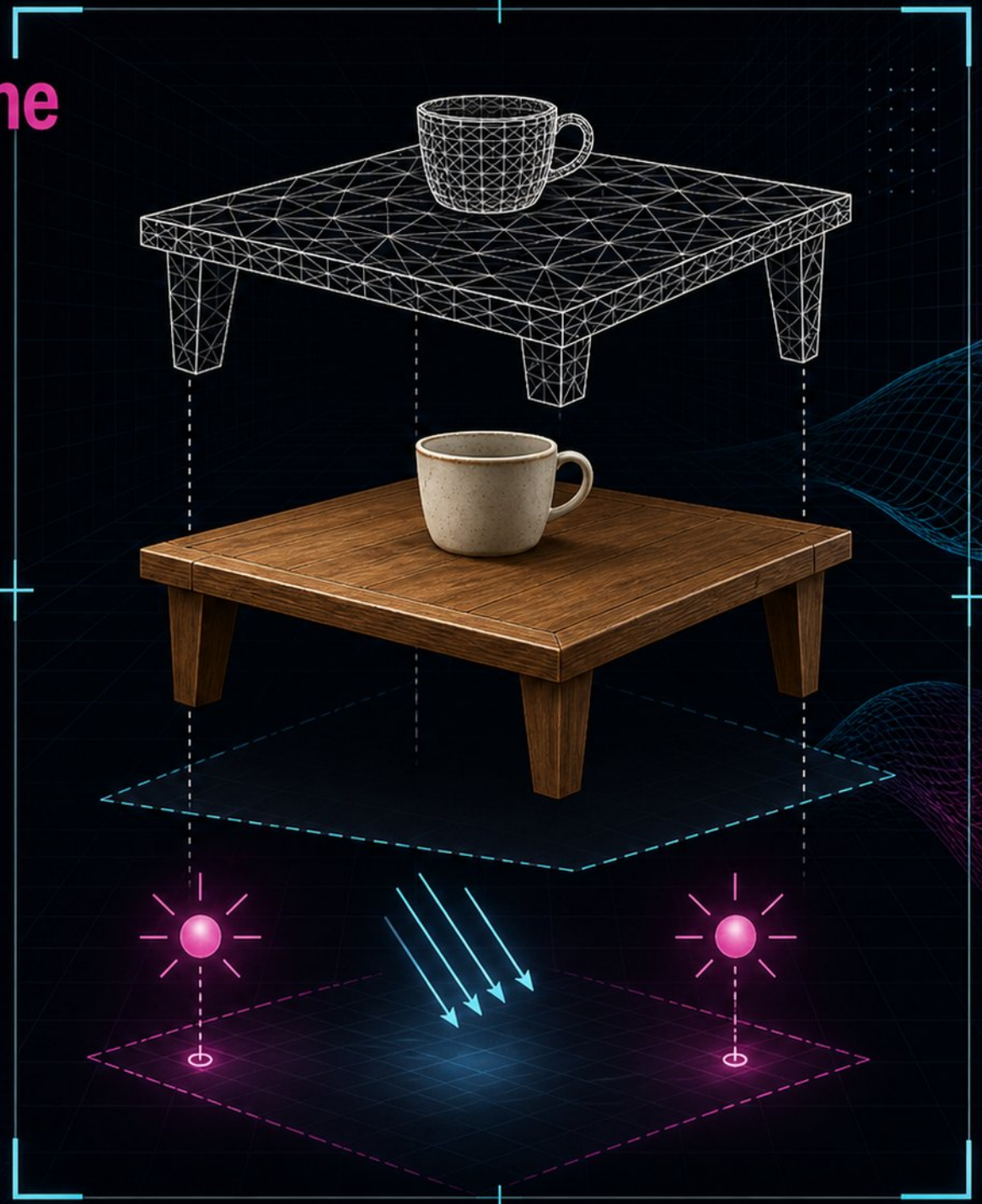
What Makes a Digital Image

- Digital images are grids of pixels with defined resolution
- Each pixel stores color using red, green, and blue (RGB) values
- Raster images are pixel grids; scenes are mathematical descriptions
- 8 bits per channel = 256 levels × 3 = 16.7 million colors
- The image is the output of a process, not just a file



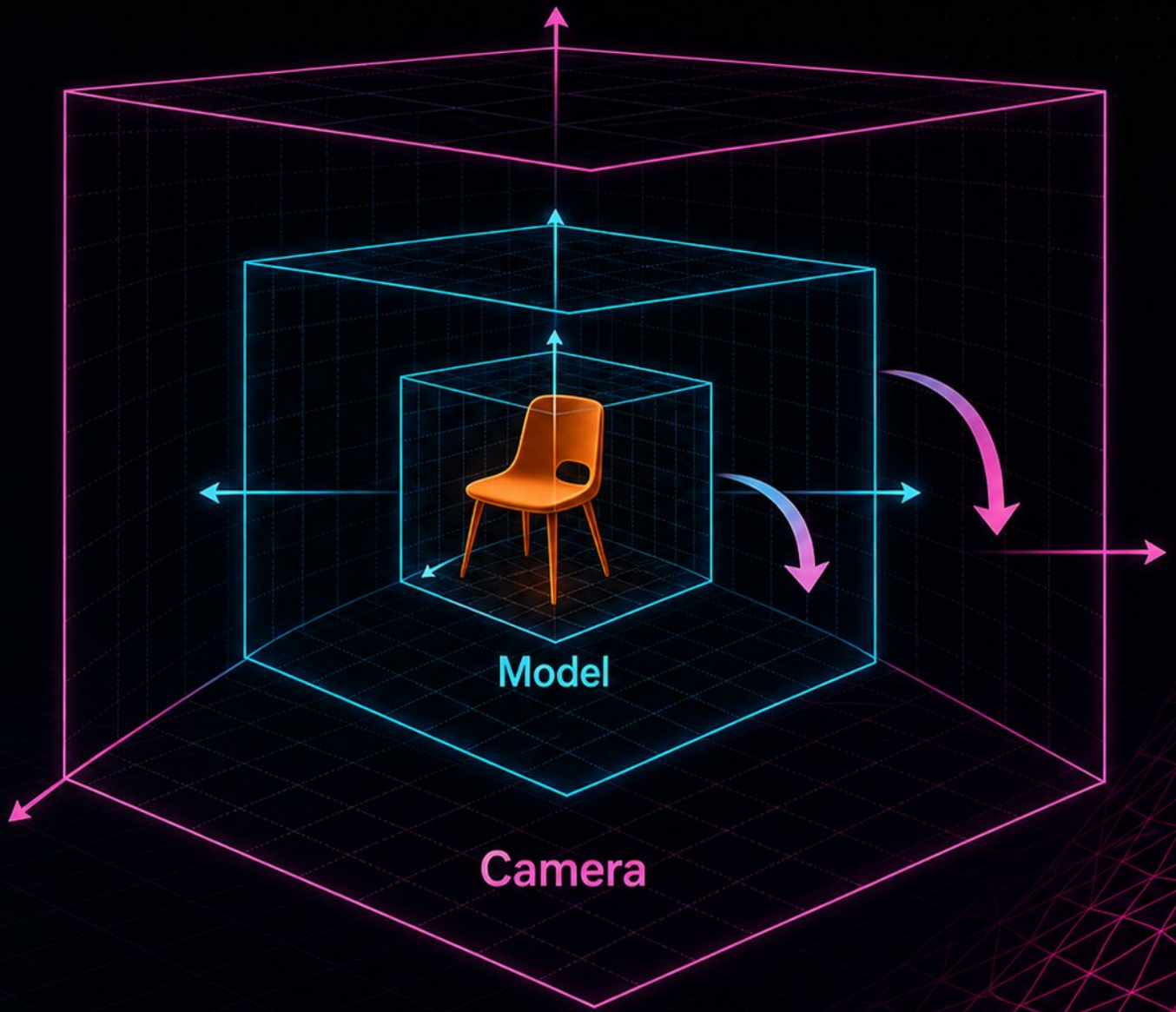
Describing a 3D Scene

- Geometry, materials, and lights are the three essential ingredients
- Meshes built from vertices and triangles form object shapes
- Materials define color, shininess, and light reflection behavior
- Point and directional lights illuminate the scene—always needed
- The scene is pure 3D data until transformed into a final image



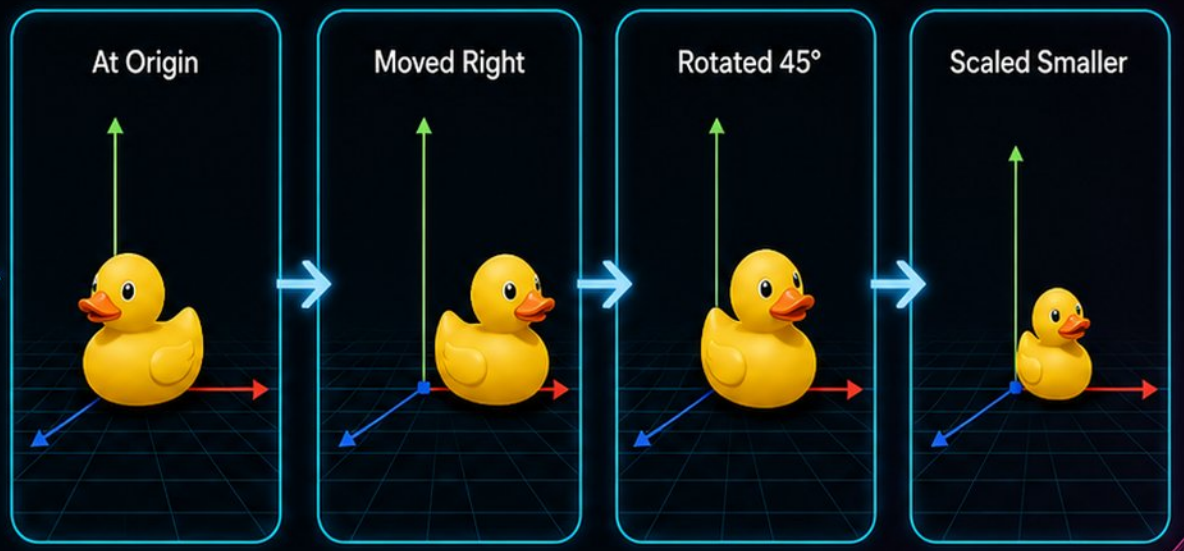
Coordinate Systems: Model, World, and Camera Spaces

- **Model space:** local coordinates relative to an object's own origin
- **World space:** shared space where all objects are placed together
- **Camera space:** the world re-expressed from the viewer's point of view
- **The pipeline:** model → world → view transforms prepare a vertex for projection

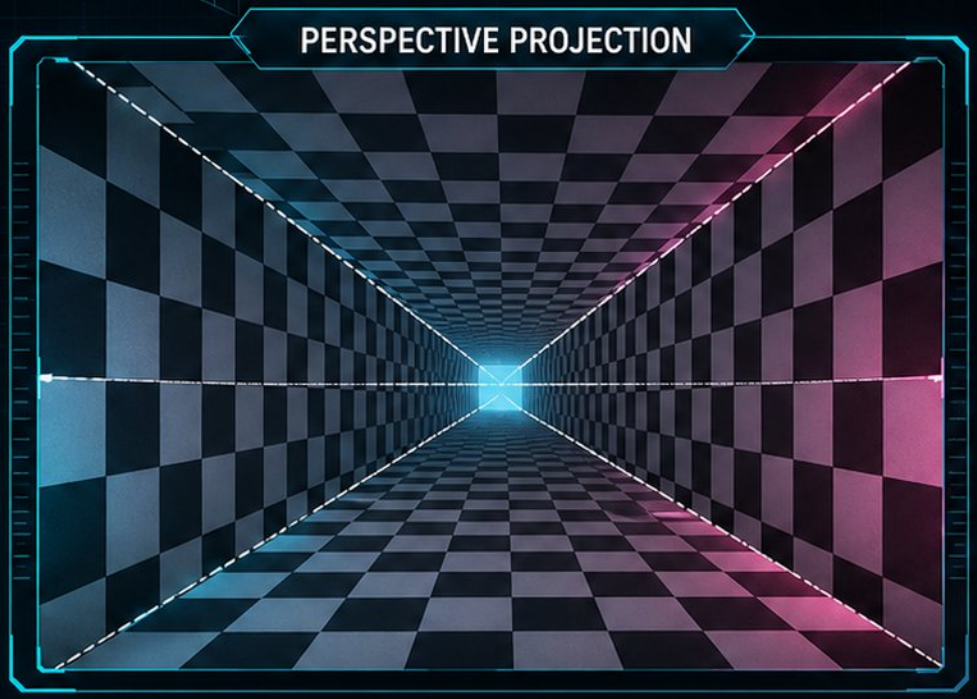


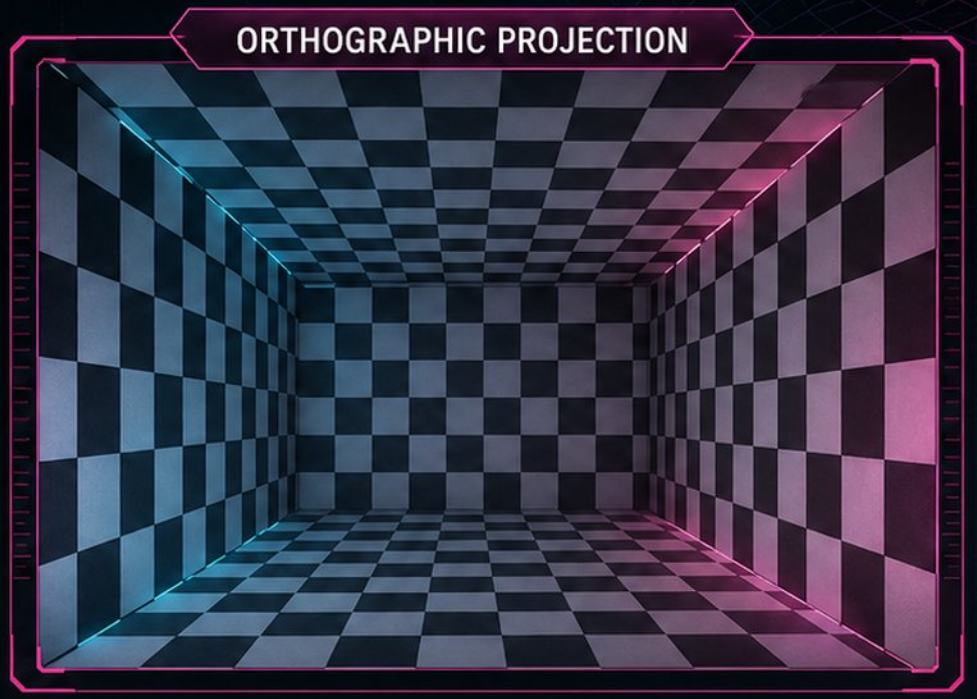
Transformations: Moving, Rotating, and Scaling Objects

- Translation, rotation, and scaling reposition objects like moving furniture
- Vertices travel: model space → world space → camera space before projection
- Order matters: turn a chair first, then place it in the room, not the reverse
- Model matrix places objects; view matrix positions the camera; together they prepare geometry
- Build intuitive spatial reasoning first—understand what happens before learning matrix math



Projection: From 3D to 2D

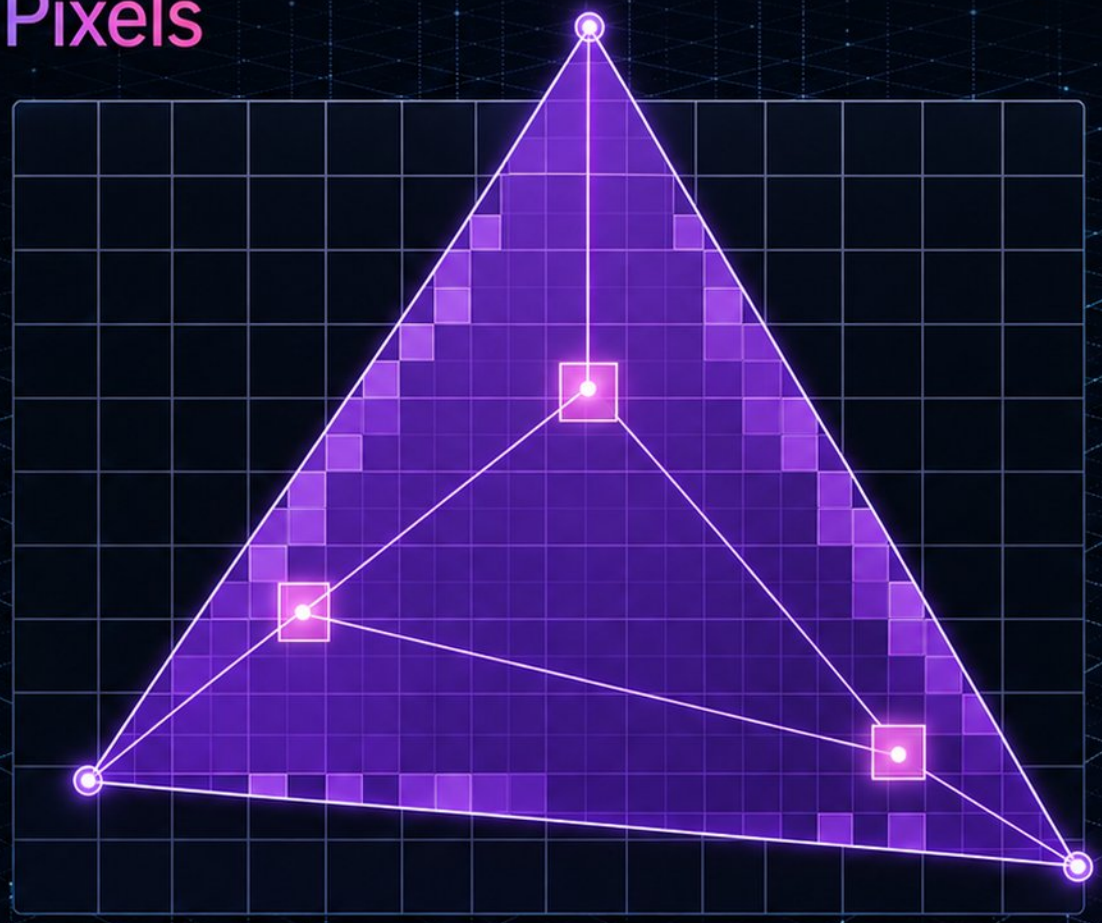




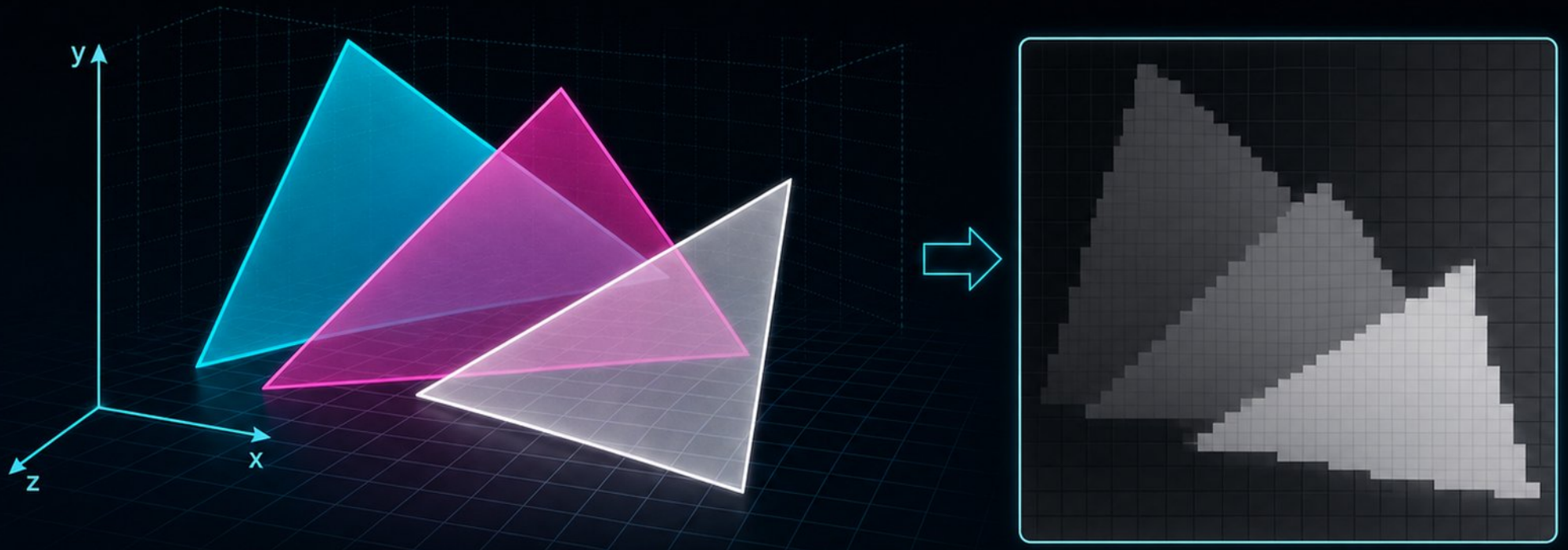
- Virtual camera defines the viewpoint via position, look-at direction, and up vector
- Perspective projection: distant objects appear smaller, parallel lines converge
- Orthographic projection keeps size constant regardless of distance — used in technical drawings
- View frustum's near plane, far plane, and field of view determine what is rendered
- Projection transforms 3D coordinates to 2D screen space while preserving depth

Rasterization:
Turning Triangles into Pixels

- Rasterization decides which pixels each projected triangle covers on screen
- Screen-space uses 2D coordinates with depth kept for every point
- Barycentric coordinates interpolate color, depth, and texture smoothly inside a triangle
- The Z-buffer stores the closest depth per pixel to resolve overlaps
- Rasterization converts continuous 3D geometry into discrete 2D fragments



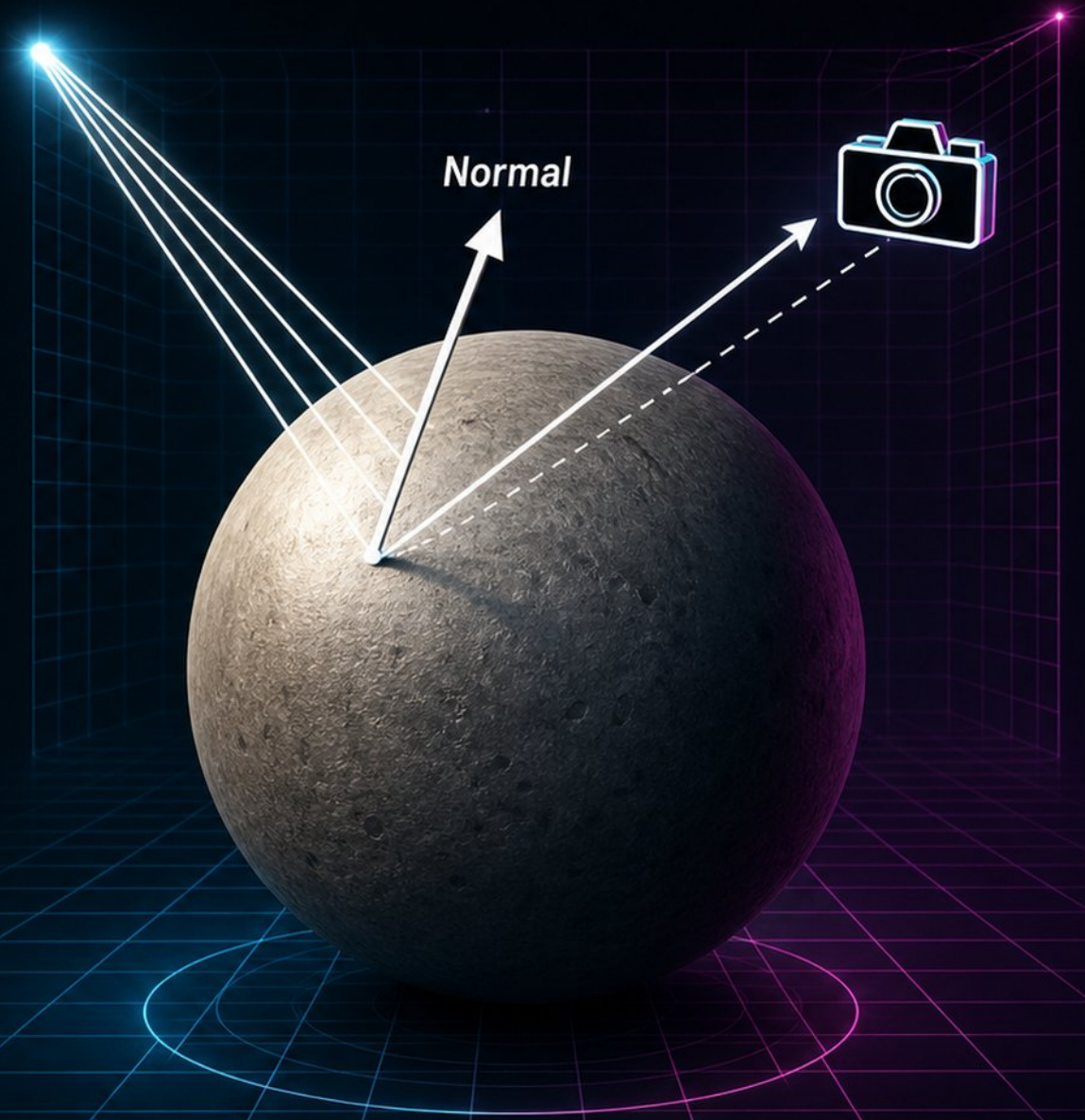
The Z-Buffer: Solving Hidden Surfaces



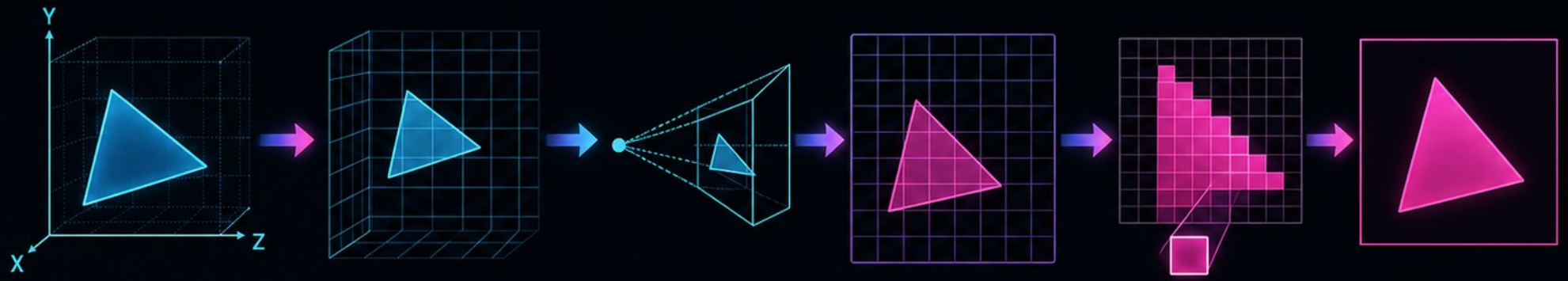
- **Hidden-surface problem:** the renderer must decide which object is visible at each pixel
- **Z-buffer** stores depth of the closest fragment per pixel in a 2D array matching screen resolution
- **Depth test:** each new fragment compares its z-value; closer fragments replace farther ones
- **Advantages:** simple, works on any polygon order, hardware-accelerated
- **Precision tip:** non-linear depth values give more precision near the camera; watch for z-fighting

Shading and the Look of Surfaces

- Pixel color: material properties + lights + surface orientation combined
- Surface normal: perpendicular arrow defining how a surface catches light
- Diffuse shading: brightness peaks when light hits the surface head-on
- Flat shading shows facets; Gouraud and Phong create smooth curved looks
- Shaders: tiny programs running per pixel to bridge scene data and final color



Putting It All Together: A Walk Through the Pipeline



3D Model
Model Space

Grid Transformation
World Space

Camera Frustum
View / Clip Space

Pixel Grid
Screen Space

Shaded Pixel
Fragment / Shade

Final Image
On Screen

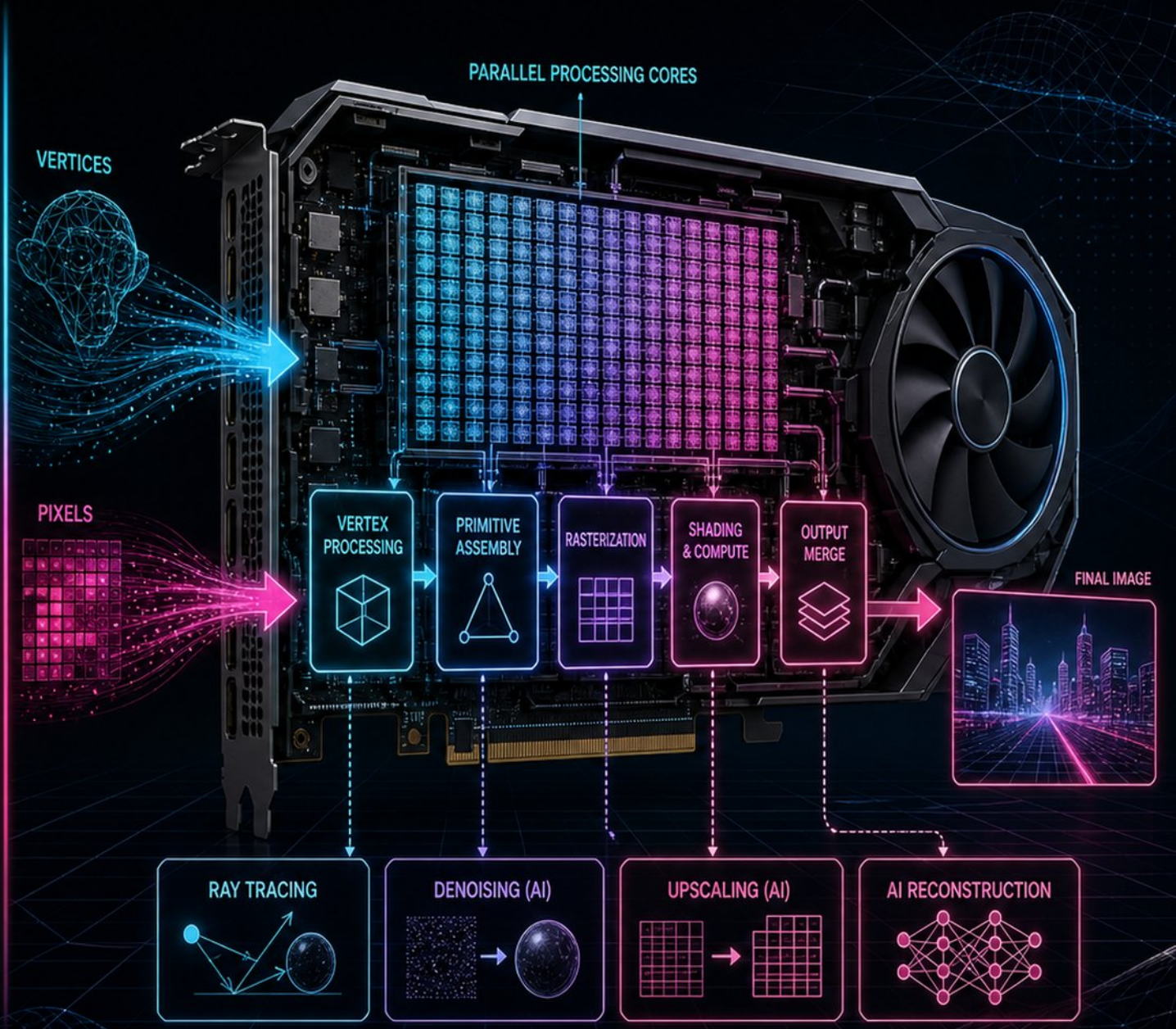
- Follow a single triangle from model space to screen pixel
- Color-coded pipeline: scene → transforms → camera/projection → rasterize → shade
- Placement, perspective, visibility, and shading all shape the result
- Watch for missing cameras, absent lights, flipped normals, or hidden geometry
- Use this pipeline as your mental model for any 3D rendering process

Real-Time vs. Offline Rendering

- "Real-time: 30–120+ FPS for games, VR, interactive apps"
- "Offline: minutes to hours per frame for movies and design"
- "Real-time relies on rasterization with optimized approximations"
- "Offline often uses path tracing for physically accurate light"
- "2026 trend: hybrid rendering with AI denoising and upscaling"

Rendering in the Real World: GPUs, Tools, and Current Trends

- GPUs are massively parallel processors with many cores working on pixels and vertices simultaneously
- Blender, Unity, Unreal Engine, and Shadertoy are free beginner tools for 3D and shaders
- Real-time ray tracing is now standard in AAA games, built into all major GPU hardware
- Neural denoising and upscaling (DLSS, FSR, XeSS) reconstruct high-quality images from fewer samples
- Path tracing in games (PRAGMATA, Resident Evil Requiem) becomes practical with AI reconstruction



From Viewer to Creator: Next Steps

- Pipeline recap: scene → transforms → camera → rasterization → shading
- Mental habit: visualize the 3D scene and camera behind every image
- Start now: Blender, Shadertoy, Unity, Unreal — all free
- Explore next: textures, PBR, animation, shader programming, ray tracing
- Broad impact: movies, games, AR/VR, UI, photography, AI imaging

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/942ffa1e/public