成均館大學校 SUNG KYUN KWAN UNIVERSITY

Graphics Systems

Computer Graphics
Instructor: Sungkil Lee

Today

- **Image formation**
- **Two basic approaches of graphics systems**
  - Physical approach
  - Pipeline approach

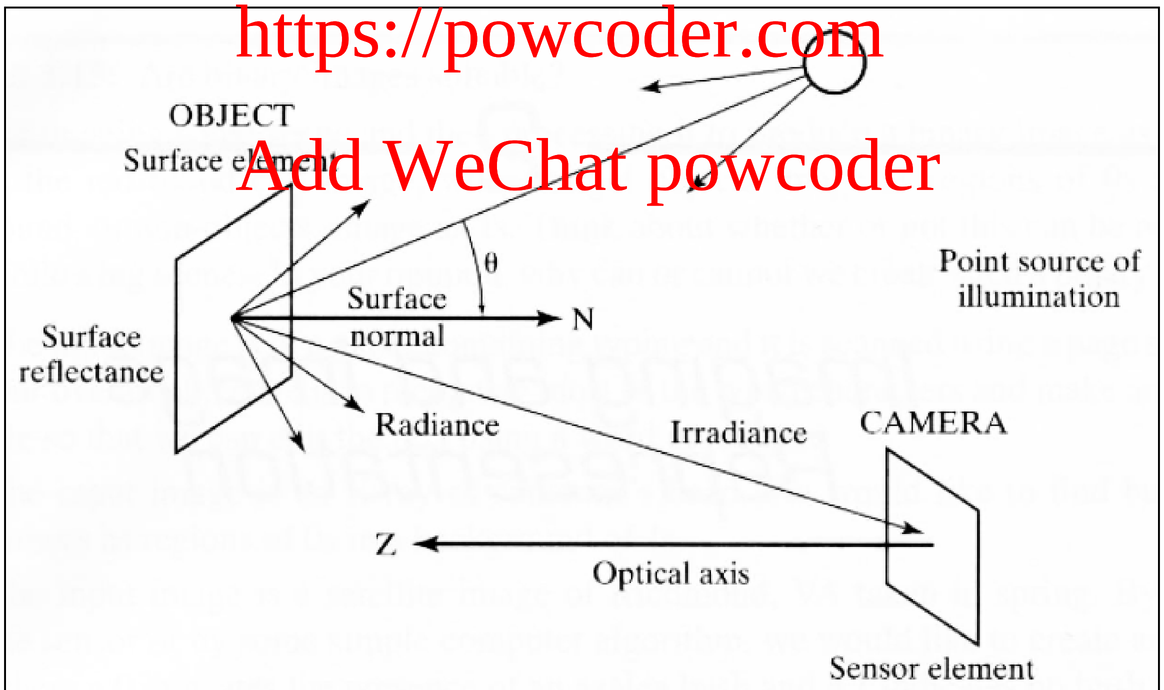
Image Formation

- **Geometry of image formation**
  - determines where the projection of a point will be located in the image plane (or the sensor plane)

- **Physics of light**
  - interaction of lights with geometric surfaces
  - determines the brightness of a point in the image plane (or the sensor) as a function of illumination and surface properties
  - ***Rendering***: simulation of light physics, yielding ***photorealism***

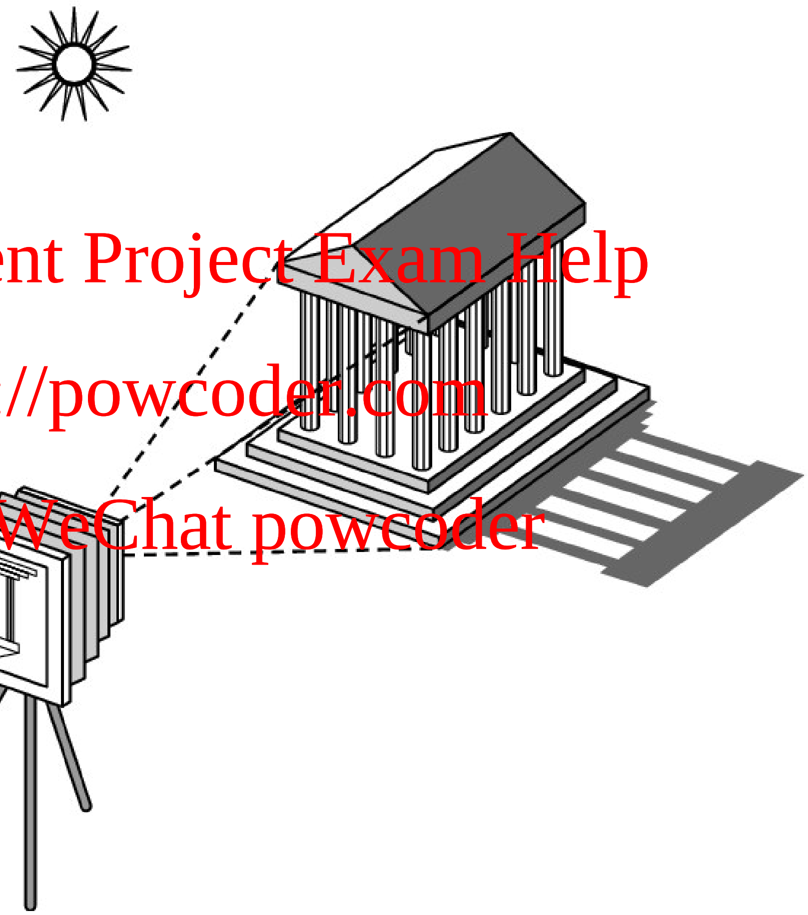
Image Formation

- **In computer graphics, we form images using a model analogous to the physical process**
  - The scene is illuminated by a single light source
  - The scene reflects radiation towards the camera
  - The camera senses it via chemicals on film.



Three Elements of Image Formation

- **Light sources**
- **Objects**
- **Camera**



(1) Light Sources

- **Light is the part of the electromagnetic spectrum that causes a reaction in our visual systems**
  - Generally visible spectra are in about wavelengths of 350-750 nm.
  - Long wavelengths appear as reds and short wavelengths as blues.

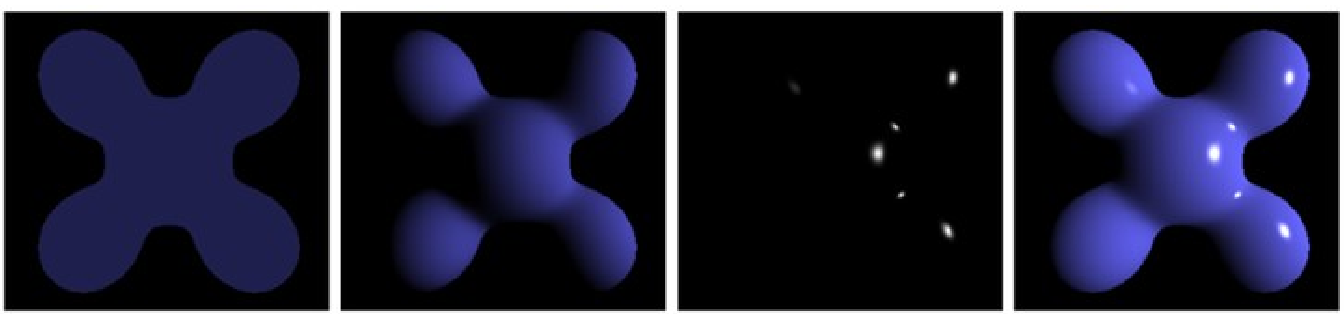
- **The typical attributes of a light source are:**

  - direction or position (often together)
  - colors (typically, white color is used)

(2) Objects

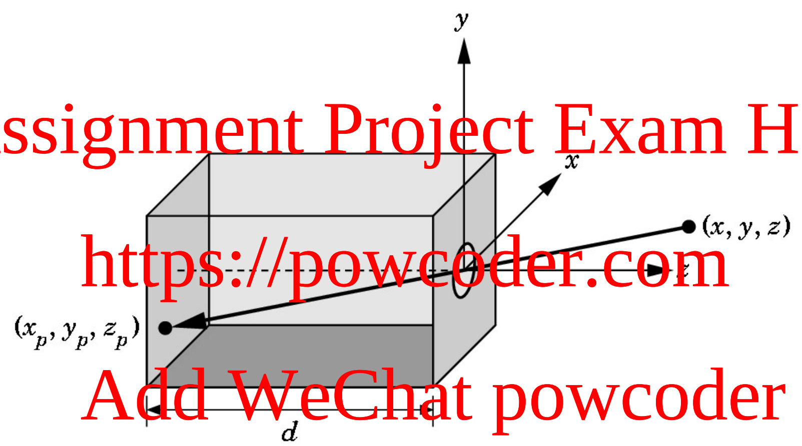
- **Objects are a set of geometries whose representation is defined mathematically.**
  - As already mentioned, vector graphics representation is used.
  - 3D positions and normal vectors are typically defined.

- **Also, surface properties of the objects are defined to simulate surface interaction with light propagation**
  - Blinn-Phong model uses ambient, diffuse, and specular colors.



(3) Cameras

- **Pinhole camera model, which causes sharp imagery, is common for most of the graphics model.**



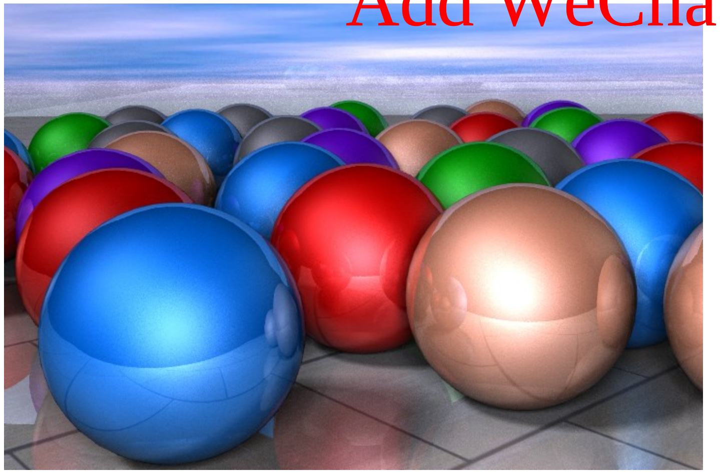
- **Typically, the following attributes define a pinhole camera.**
  - 3D transformation of a camera
  - Viewing angle, the aspect ratio of the sensor size, the range of object depths

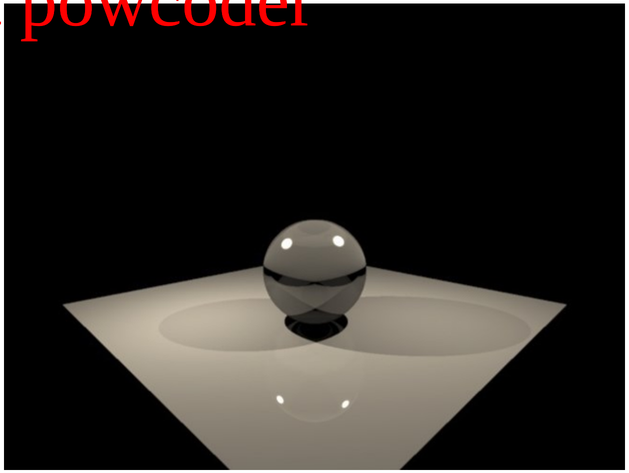
Graphics Systems

Physical Approach

- **Global illumination**
  - Captures all the light inter-reflections among the surfaces and light sources
  - Usually implemented on software
  - Very slow and suitable for high-quality film production

- **Typical example: ray tracing**

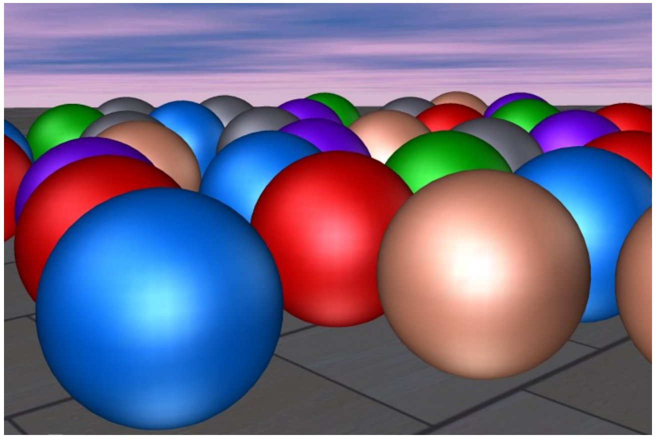




Pipeline Approach

- **Local/direct illumination**
  - Captures only direct light-object reflection
  - Based on rasterization
  - High performance suitable for real-time interactive rendering
  - However, quality is degraded with significant approximations.

- **Typical examples**
  - OpenGL, on which this course focuses, and DirectX
  - Facilitated by special-purpose graphics hardware (GPU)



Pipeline Approach

- **Missing visual effects in local illumination model**
  - Inter-object reflections
  - Refractions
  - Shadows

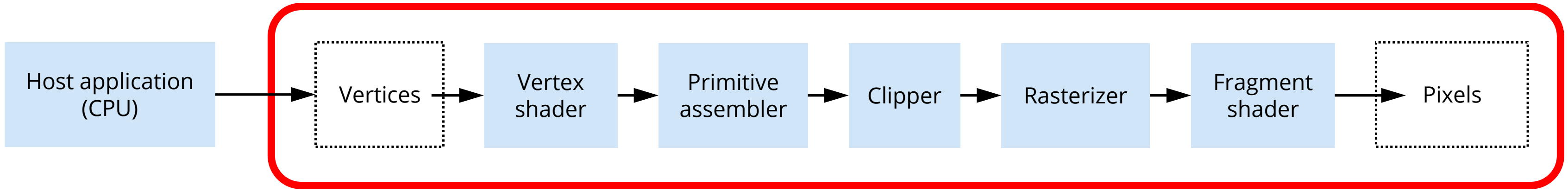
- **However, most of the real-time rendering techniques simulate such effects through approximation.**
  - In most cases, visually plausible but physically degraded.

Pipeline Approach

Pipeline Approach

- **Process objects one at a time in the order they are generated by the application**
  - One unit independently processes a single object but there are more units processing more objects at the same time.
  - Local/direct illumination can be computed without object dependency, and thus, objects are processed independently

- **Pipeline architecture on graphics hardware**



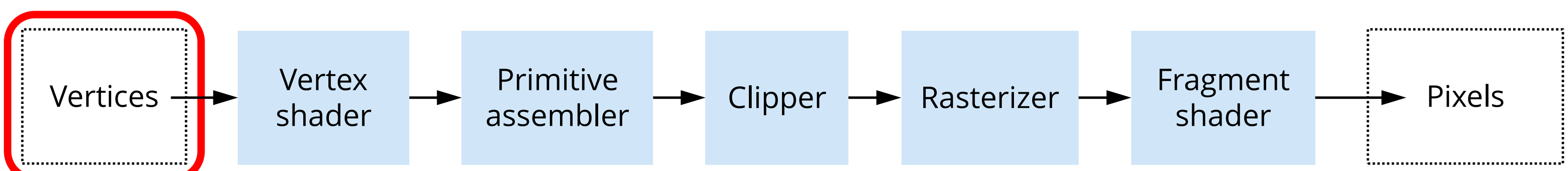
Graphics Pipeline (GPU)

Before Vertex Processing

- **A host application transfers the data in main memory to the GPU memory**
  - Data in GPU memory is only the copy of ones in main memory.
  - We need to maintain the source of GPU memory.

- **Vertex data (buffer) are transferred to GPU.**

  - These do not have to be done for every rendering frame.
  - When there are changes, we update GPU-side data by copying them.

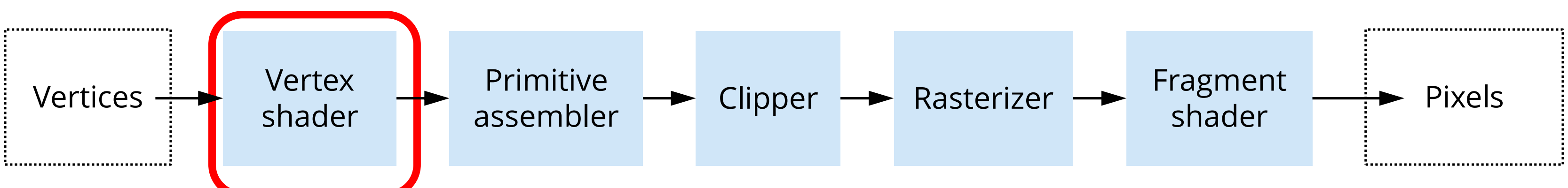


Vertex Processing: 3D Transformation

- **Vertex indicates a single 3D point with its attributes**
  - 3D position, normal vector, and texture coordinate

- **Primary role of vertex processing is positioning a single vertex**
  - Local object coordinates → world object coordinates
  - World object coordinates → camera (eye) coordinates

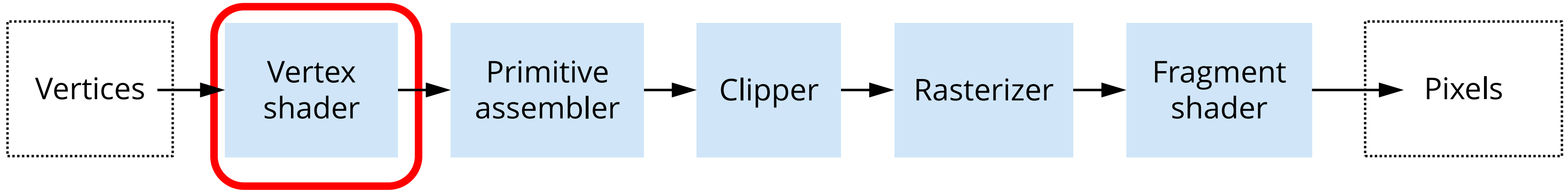
- **Every change of coordinates is equivalent to a 4×4 matrix transformation**



Vertex Processing: Projection

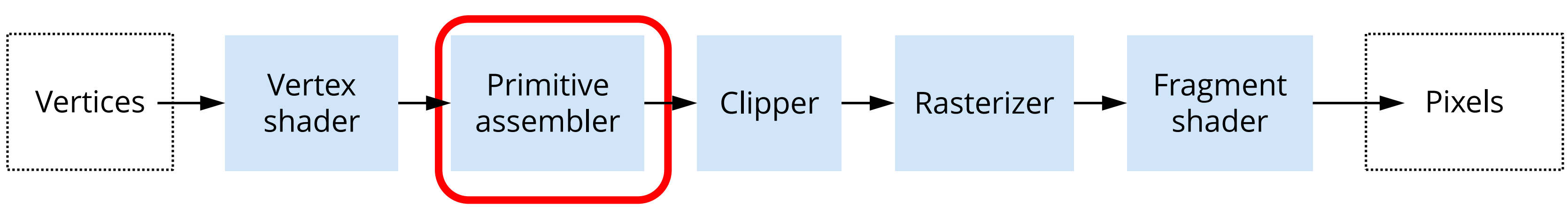
- ***Projection*** **is the process that projects 3D camera coordinates to 2D screen (window) coordinates**
  - Perspective projection
    - all projectors meet at the center of projection
  - Parallel projection:

- **The projection is also done with a 4×4 matrix multiplication.**



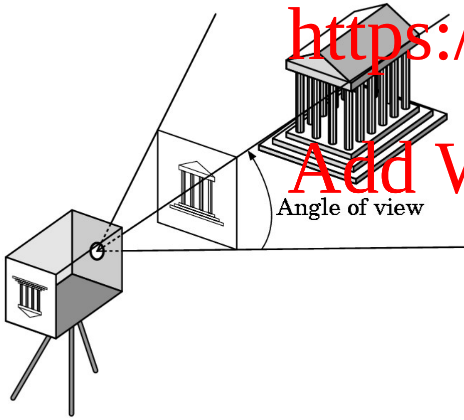
Primitive Assembly

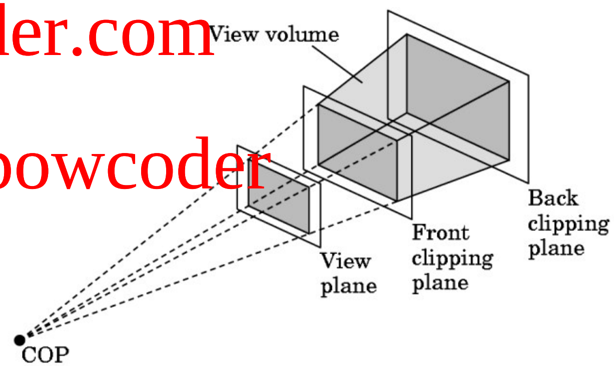
- **Vertices must be collected into geometric objects prior to later steps**
  - Line segments: 2 vertices
  - Triangles: 3 vertices

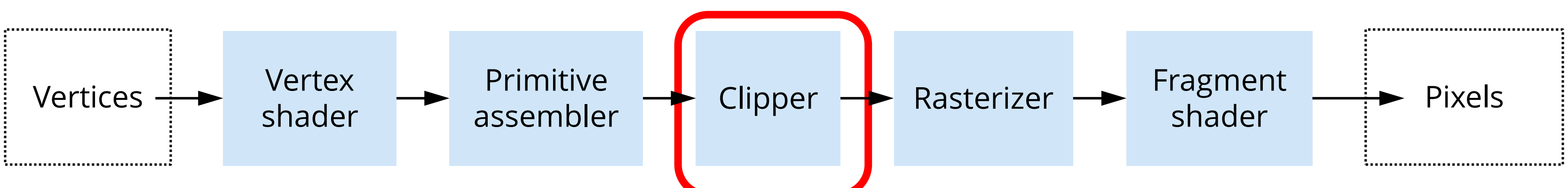


Clipping

- **As a real camera cannot see the whole world, the virtual camera can only see part of the world.**
  - Invisible objects outside view volume are said to be ***clipped***.
  - Clipped triangles are no more processed in later steps.

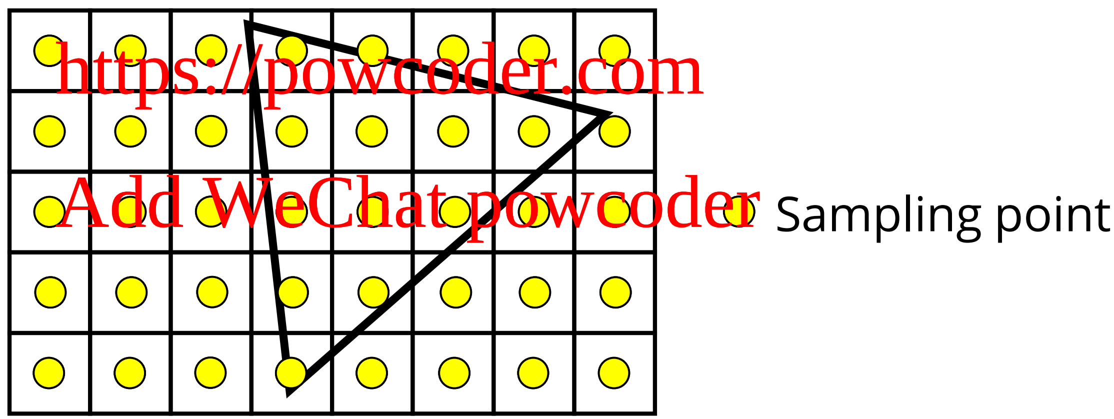


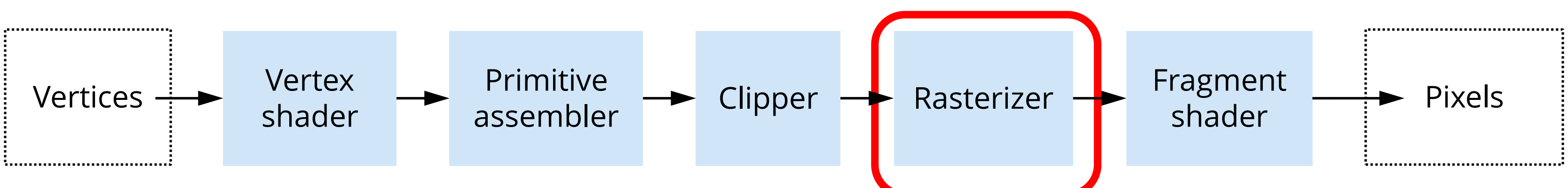




Rasterization

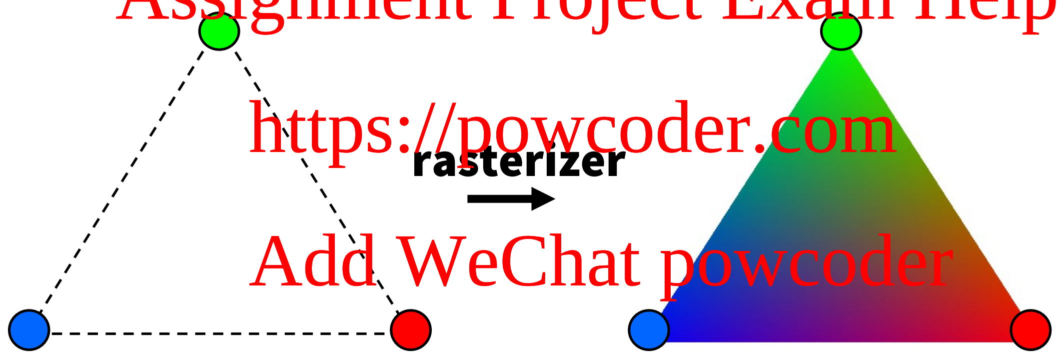
- **Rasterization**
  - Conversion of non-clipped objects (in vector graphics formats) to potential pixels (called the *fragments*).
  - Produce a set of fragments whose centers lie inside in each triangle.





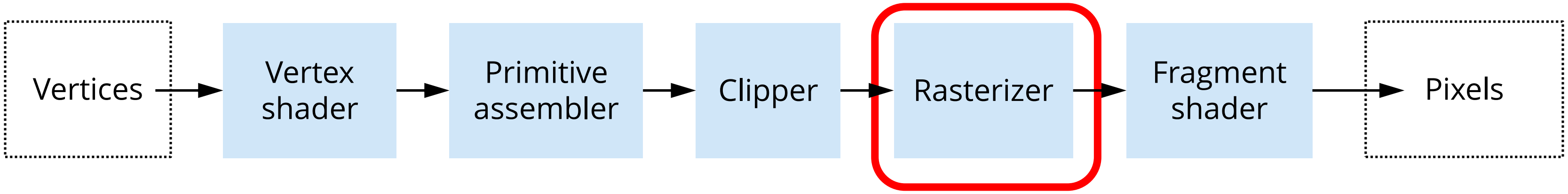
Rasterization

- **Vertex attributes are interpolated over objects by the rasterizer.**
  - 2D screen position, normal vectors, texture coordinates
  - Color and depth attributes



3 vertices in a triangle with RGB colors

Rasterized fragments with interpolated attributes

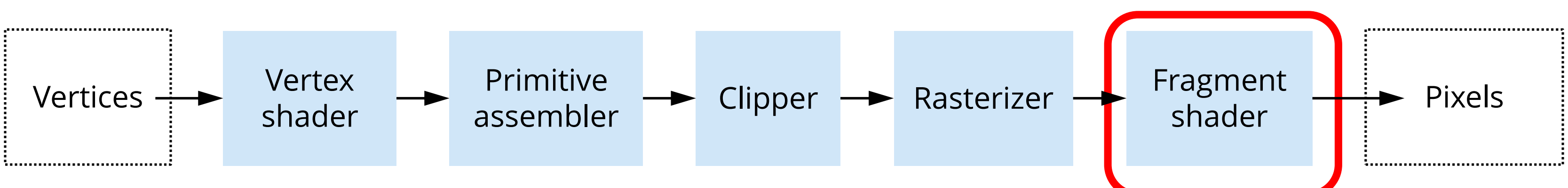


Fragment Processing

- **Fragments are processed to *determine the color* of the corresponding pixel in the frame buffer**
  - Colors can be determined by texture mapping or interpolation of vertex colors

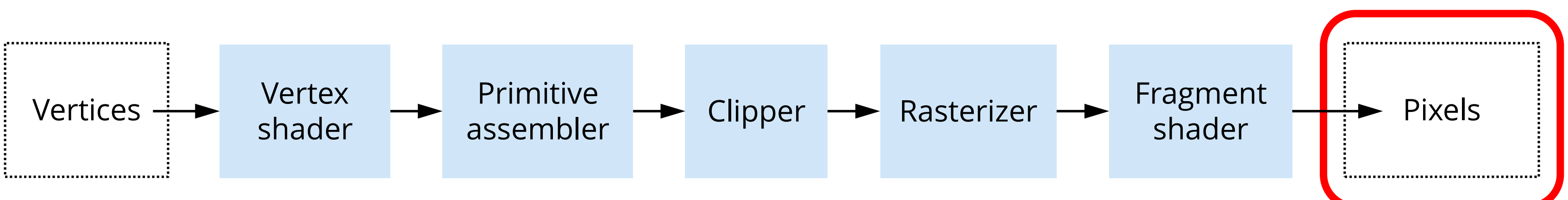
- **Hidden-surface removal using *depth buffering***

  - Fragments may be blocked by other fragments closer to the camera
  - Additional frame buffer, called the ***depth buffer***, determines whether the current fragment is nearer than one in the frame buffer.

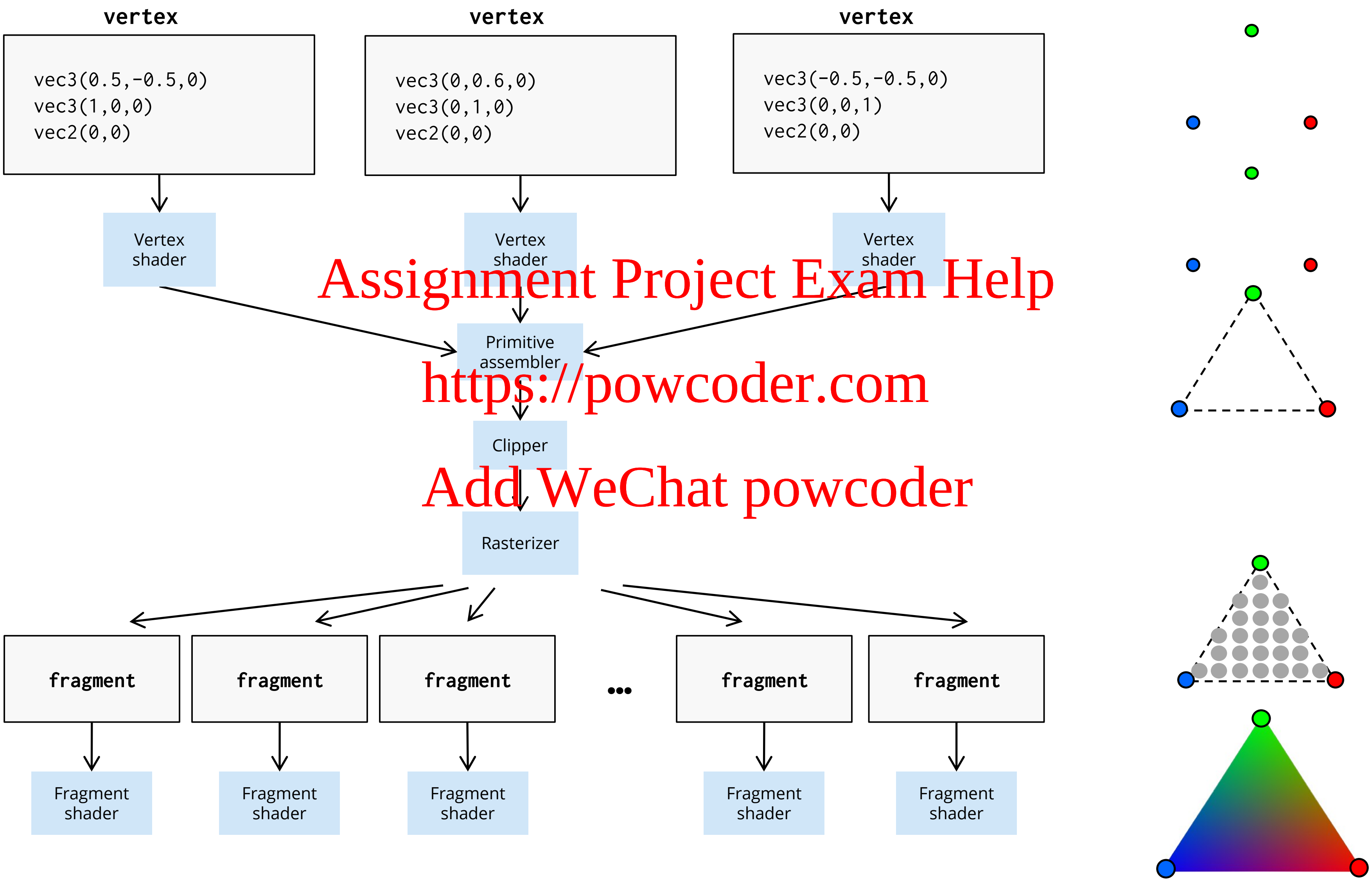


Display

- **Still alive fragments (now, we call pixels) are transferred to the display devices.**
  - Now we can see an image in the monitor.



Example Data Flow

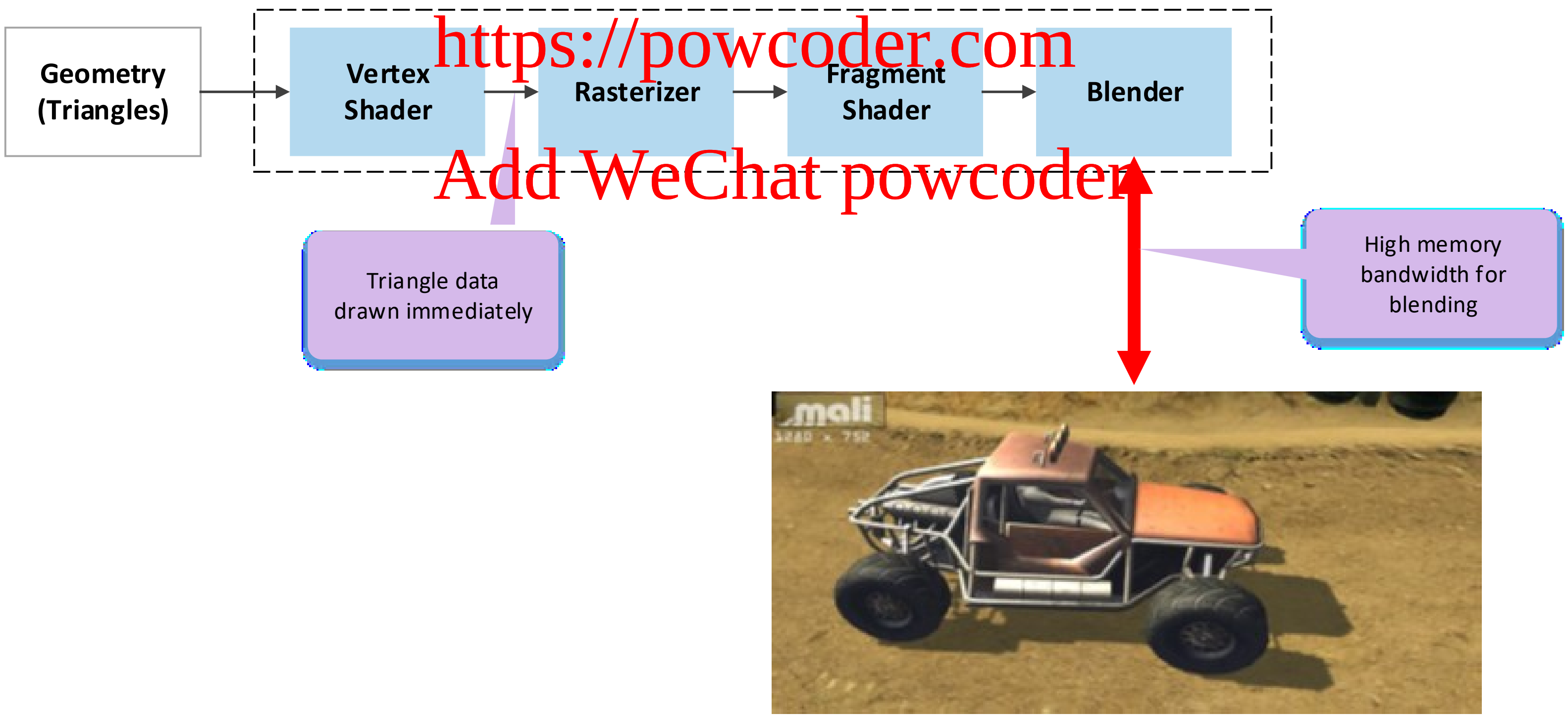


Appendix:

Tile-Based Rendering for Mobile Graphics

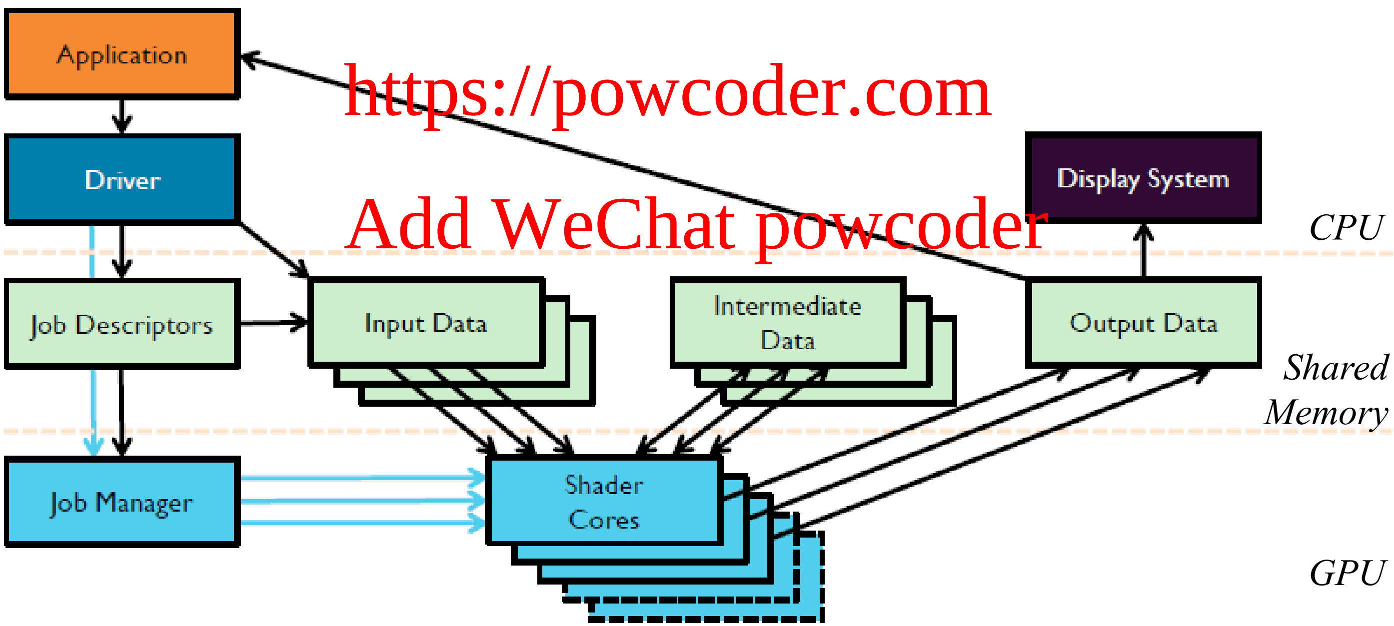
Immediate Mode Rendering (IMR)

- **Background**
  - IMR (typical desktop-like rendering pipeline with the full framebuffer) costs large bandwidth/space and power consumption.
    - c.f., IMR here is different from IMR in desktop rendering (IM vs. Retained Mode)
  - Mobile devices are limited in physical space and power consumption.



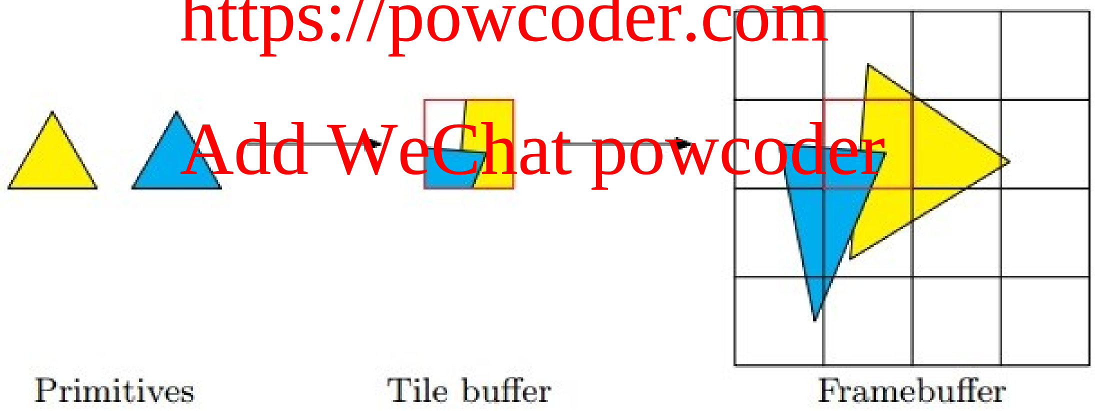
Immediate Mode Rendering (IMR)

- **Background**
  - IMR needs costly update (e.g., blending and frame buffer operations) with intermediate data.
  - e.g., basic data flow in ARM



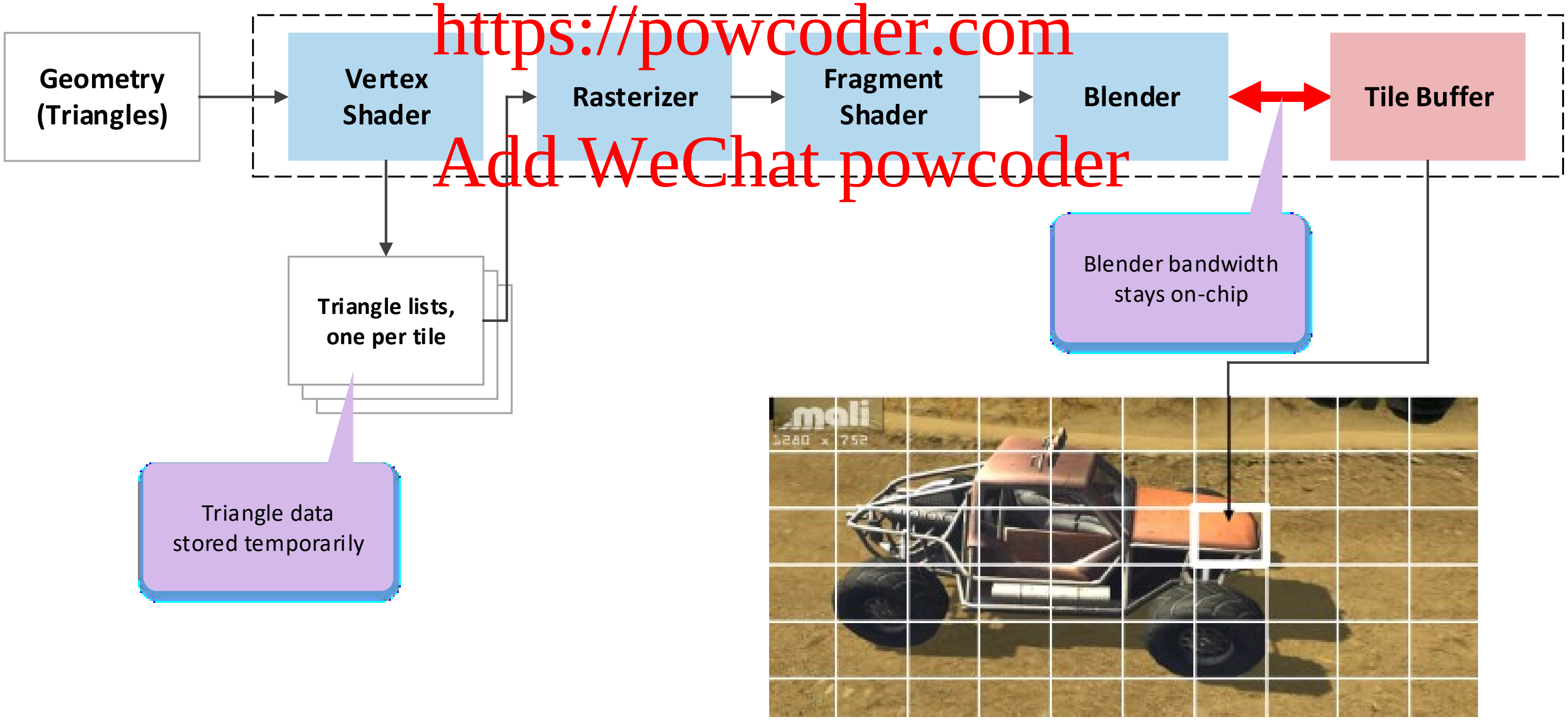
Tile-Based Rendering (TBR)

- **Tile-Based Rendering**
  - Subdivide scenes into smaller **tiles** (e.g., 16x16 or 32x32) in screen space and render each section of tile separately.
  - Intermediate data interact with a **small** and **on-chip** (local) tile buffer, and thereby, memory bandwidth is significantly reduced.



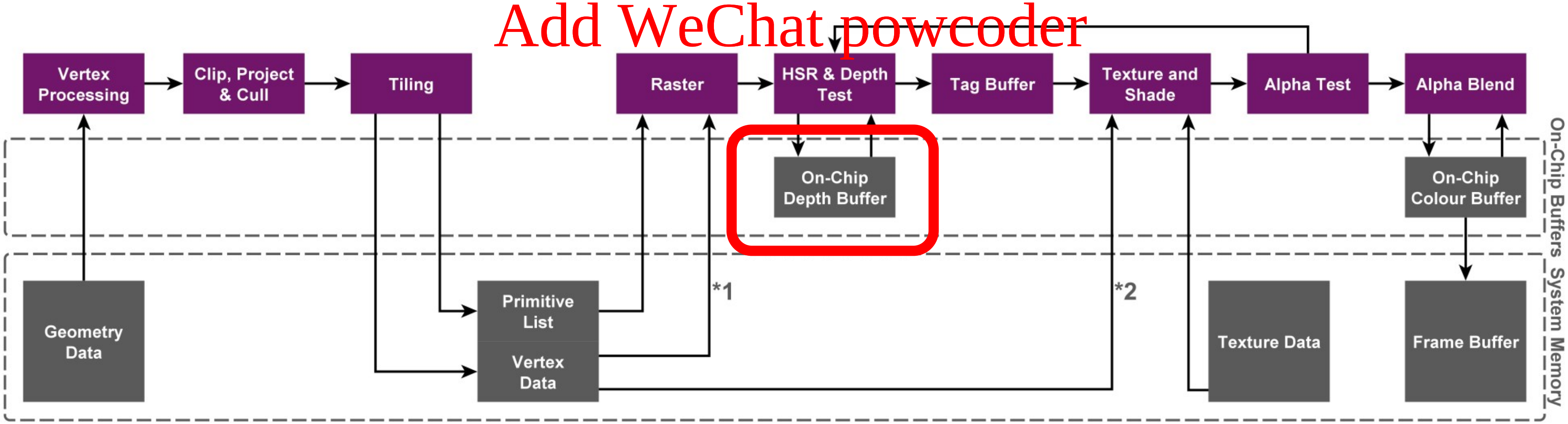
Tile-Based Rendering (TBR)

- **Tile-Based Rendering**
  - Triangles are not directly sent to a rasterizer, but sorted by their location (i.e., tile ID) in the middle of the graphics pipeline.
  - When their tile is activated, the triangles start to be rasterized.
  - Temporary triangle lists are required, but not too large in mobile rendering.



Tile-Based Deferred Rendering (TBDR)

- **TBDR (mostly in PowerVR)**
  - Rasterization is **deferred** until all the primitives are stored into the tile triangle lists.
  - To gain additional speedup, the triangles are sorted front-to-back in advance to facilitate early-Z (pre-raster hidden surface removal).
    - This step uses on-chip (tile) depth buffer.
    - Made more efficient, combined with on-chip color blending (in TBR).



Tile-Based Deferred Rendering (TBDR)

- **TBDR (mostly in PowerVR)**
  - After the hidden surface removal (HSR), the pixel shading starts (with texture fetch). In other words, rendering (more precisely, texturing and shading) is deferred until after a per-pixel visibility test is passed.