Computer Graphics - TAU

Moab Arar

Rendering

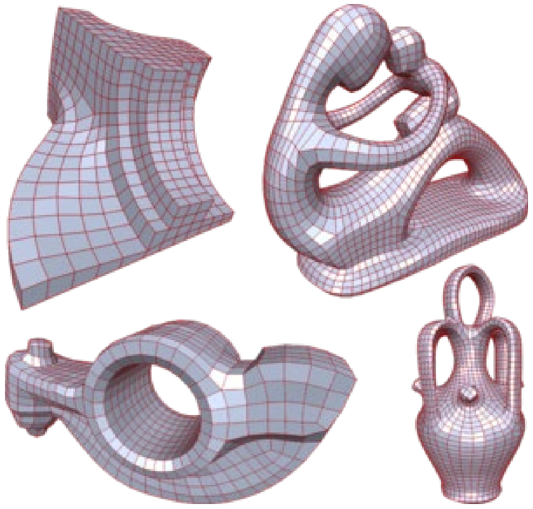
- The process of turning 3D virtual world into 2D images:
  - Vidoe games
  - Animation Movies
- What rendering algorithms do you know ?
  - Ray Tracing and Ray Casting
- What are the advantages/disadvantages of Ray Tracing ?
- Today we are going to learn about Graphical Pipeline

Graphical Pipeline:

- Set of steps need to be taken in order to turn 3D scene into 2D image
  - The graphical pipeline is conceptual model
  - Highly dependent on the underlying available Software and Hardware accelerators
- The model of graphical pipeline is usually used in real-time rendering
  - Each step is backed with efficient algorithms that are usually hardware accelerated (e.g., using GPUs)

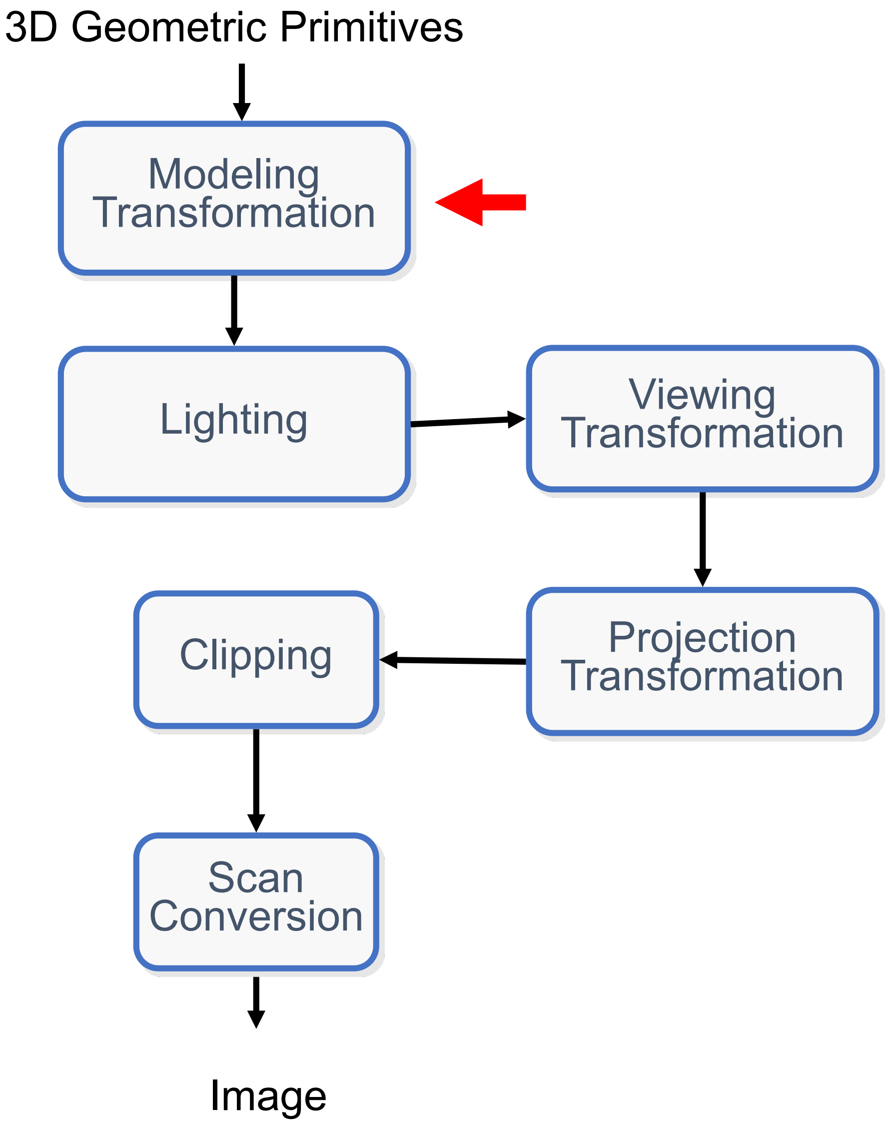
Motivation

- How do you represent 3D surfaces ?
- Maybe: using Implicit representation ?
  - Not tractable for complex shapes
- A 3D mesh (usually triangles):
  - Simple objects can be composed of thousands of triangles!



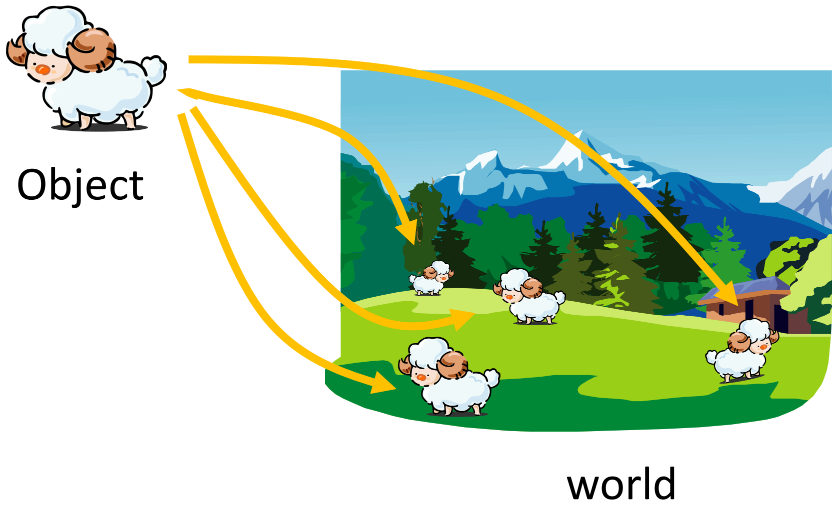
The Graphical Pipeline

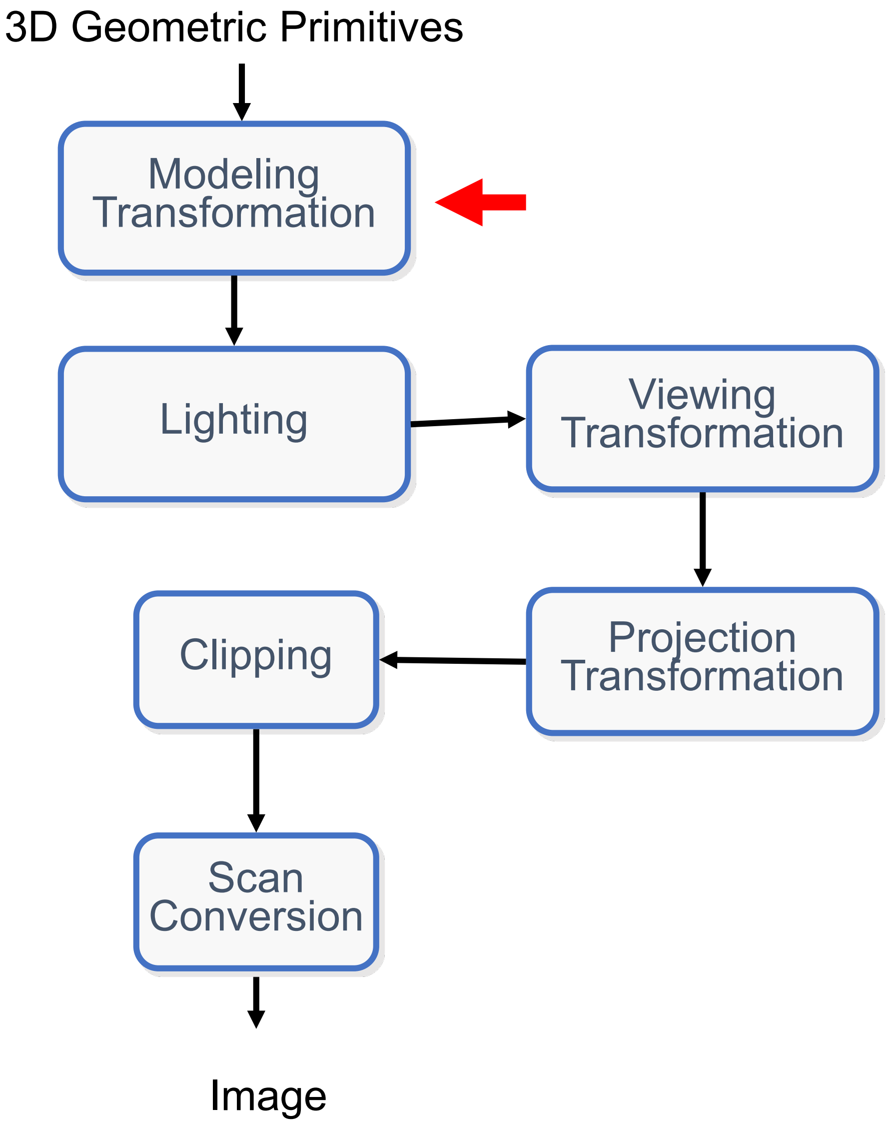
- Step 1 - Modeling Transformation
  - 3D models (assets) are usually created speartaley.
  - Each model is designed in its local coordinate system – benefits:
    - Symmetry
    - Scale independent
    - Reusability
    - And more
  - 3D scences on the other hand are composed of many 3D objects:
    - Objects are located in the **world coordinate system**



The Graphical Pipeline

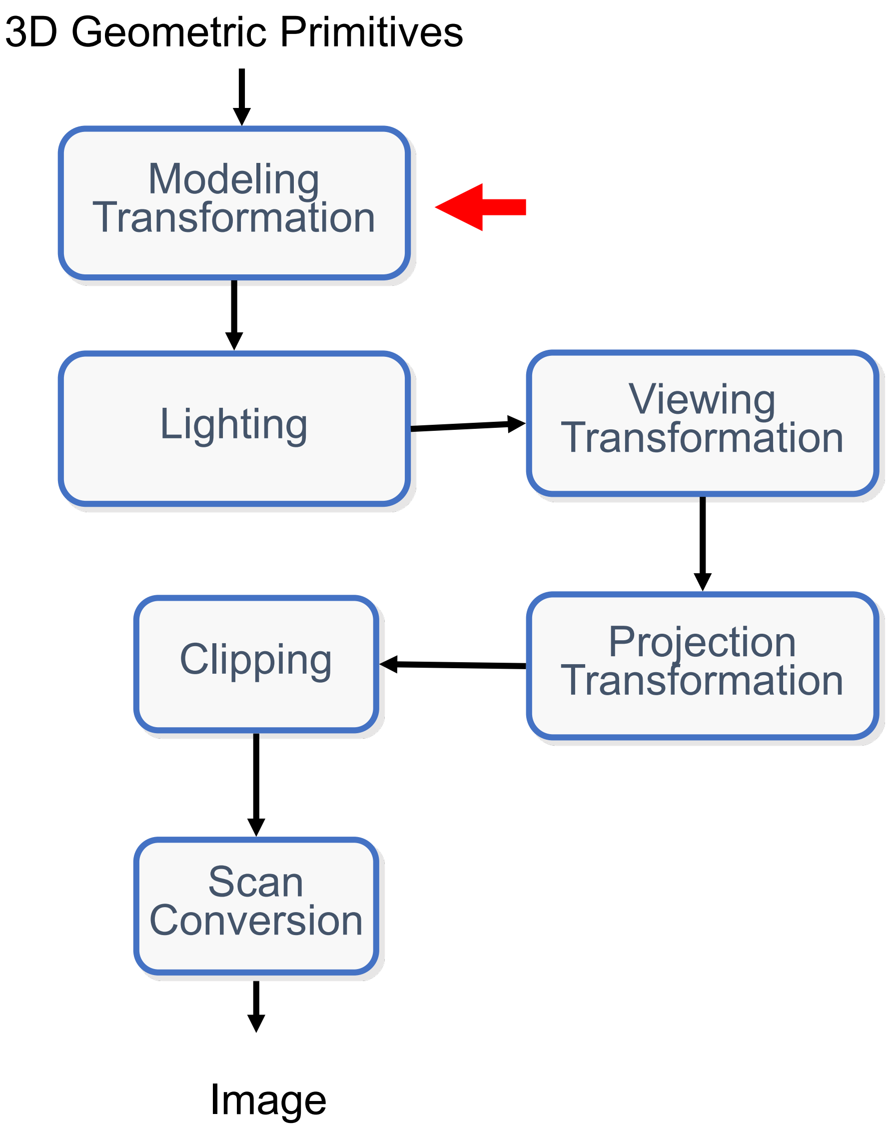
- Step 1 - Modeling Transformation
  - Each model is transformed from its local coordinate system to the world coordinate system:





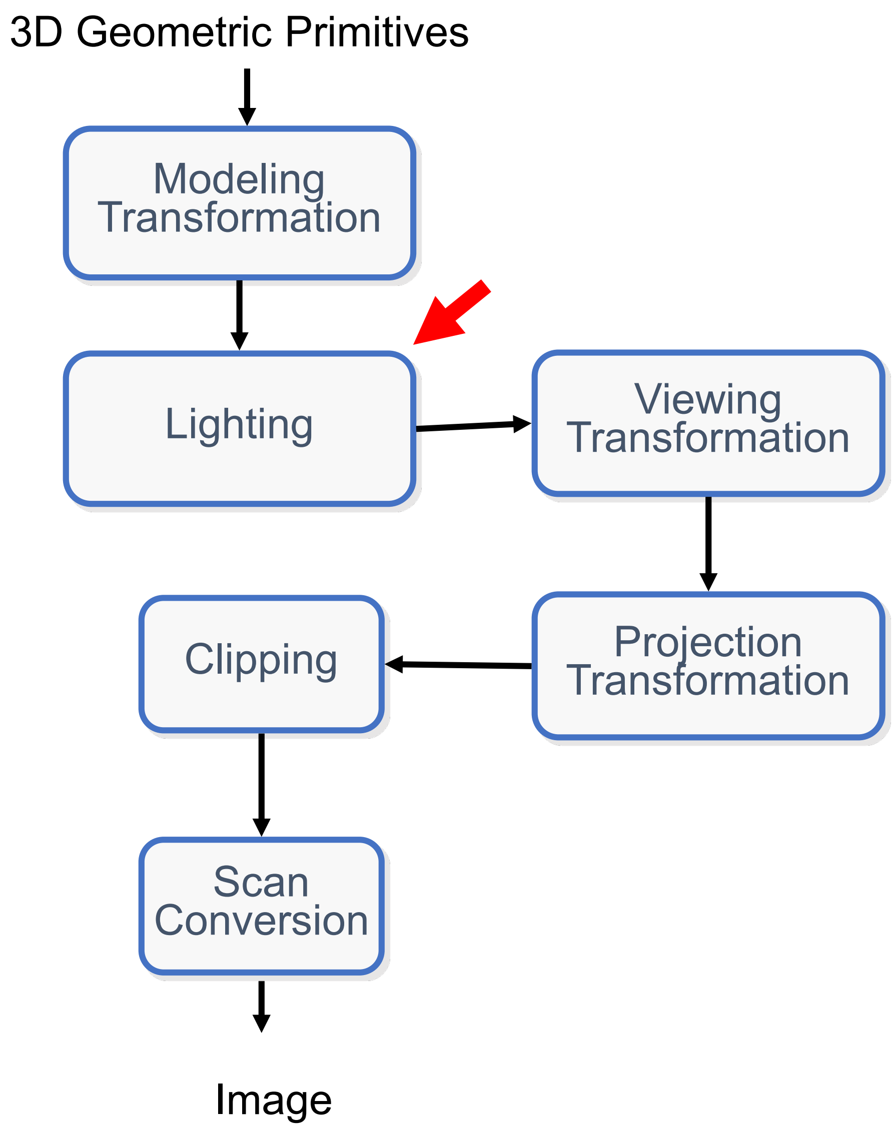
The Graphical Pipeline

- Step 1 - Modeling Transformation
  - This step is implemented by multiplying each vertex of the 3D model by a (different) model transformation matrix (GPUs allow parallel compute of the matrix multiplication).



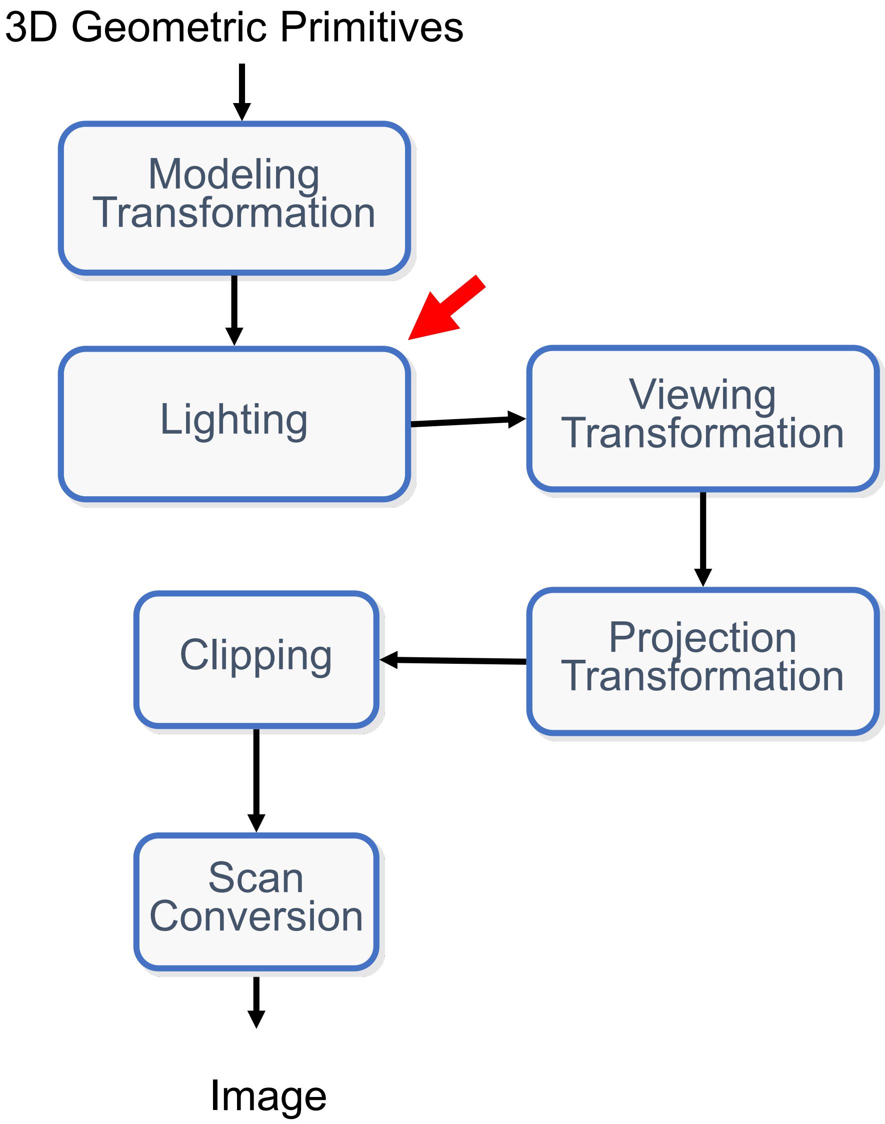
The Graphical Pipeline

- Step 2 – Lighting
  - Why lighting is important ? It is what helps us perceive depth in images



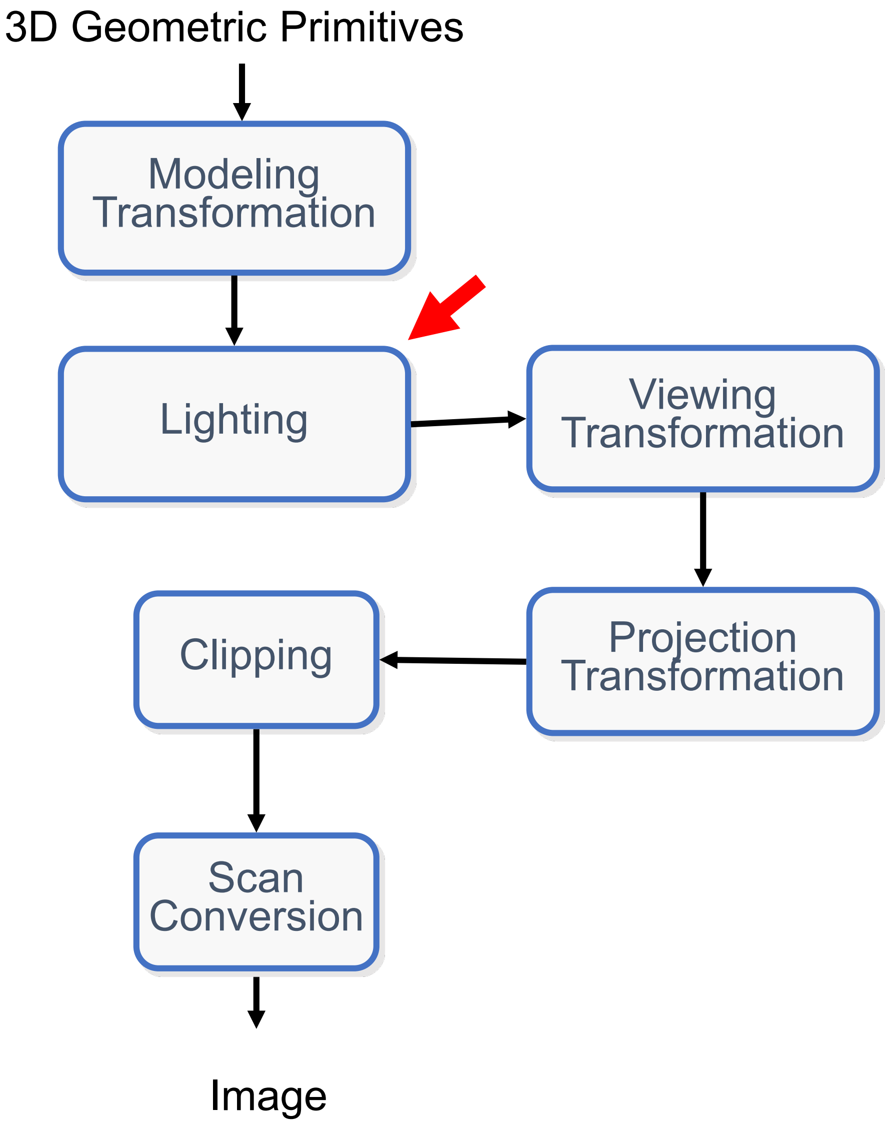
The Graphical Pipeline

- Step 2 – Lighting
  - Why lighting is important ? It is what helps us perceive depth in images



The Graphical Pipeline

- Step 2 – Lighting
  - What do we need to support lighting (aka shading):
  - Light Properties:
    - What type of light sources exists (e.g. point light, directional light, spotlight)
    - Light entensity and color
    - Light location/direction
  - Material Properties
    - How light interacts with the 3D object (diffuse and specular reflection)

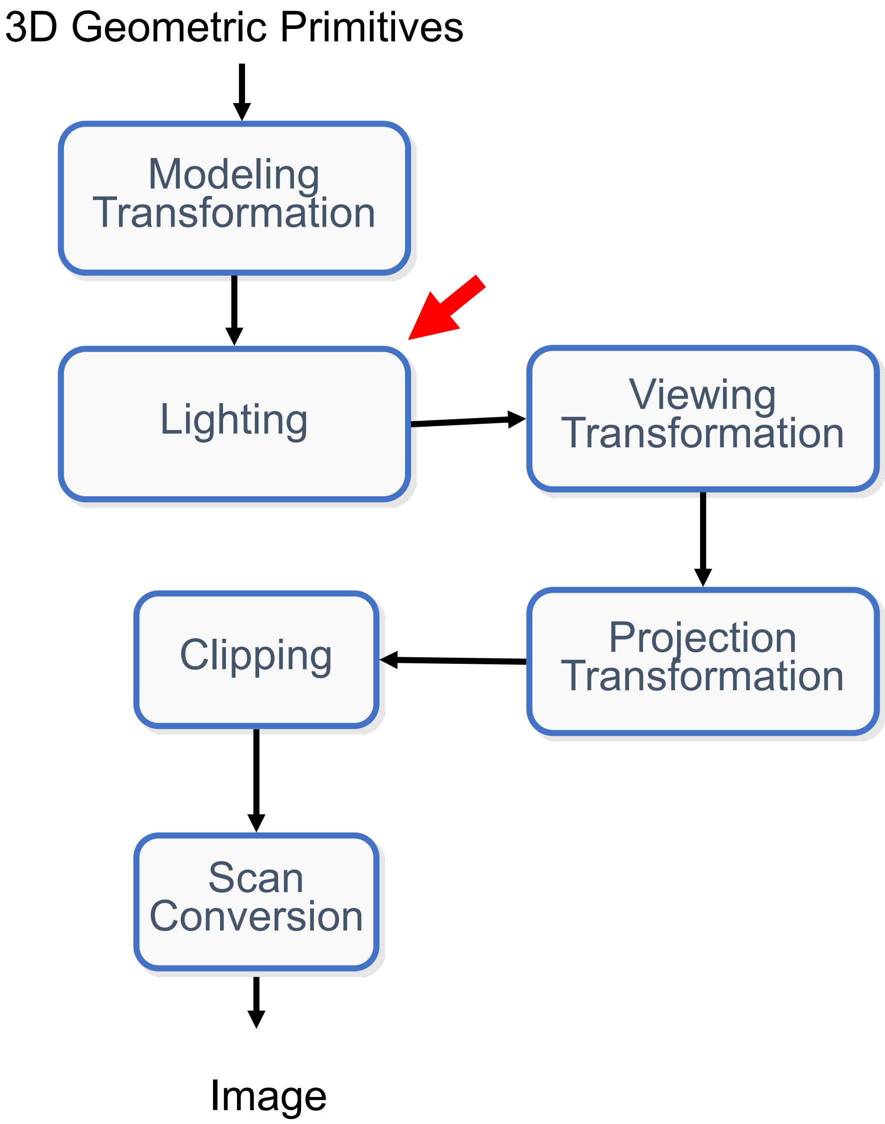


The Graphical Pipeline

- Step 2 – Lighting
  - Remember the Phong reflectance model from Ray-Tracing ?

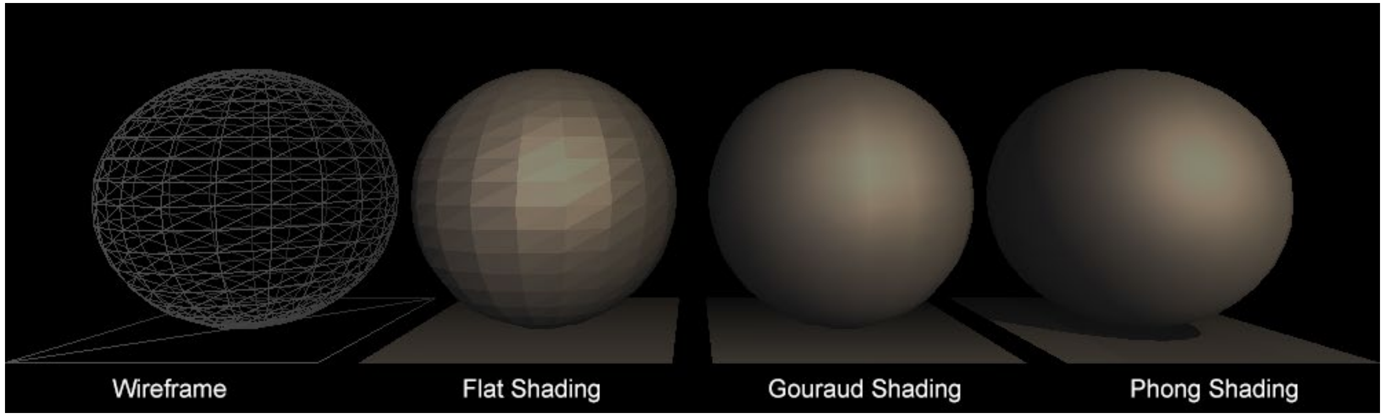
$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i)^n I_i)$$

  - How can we use this Formula to determine the color of the triangle we want to draw (remember, objects are usually represented as triangle-meshes)

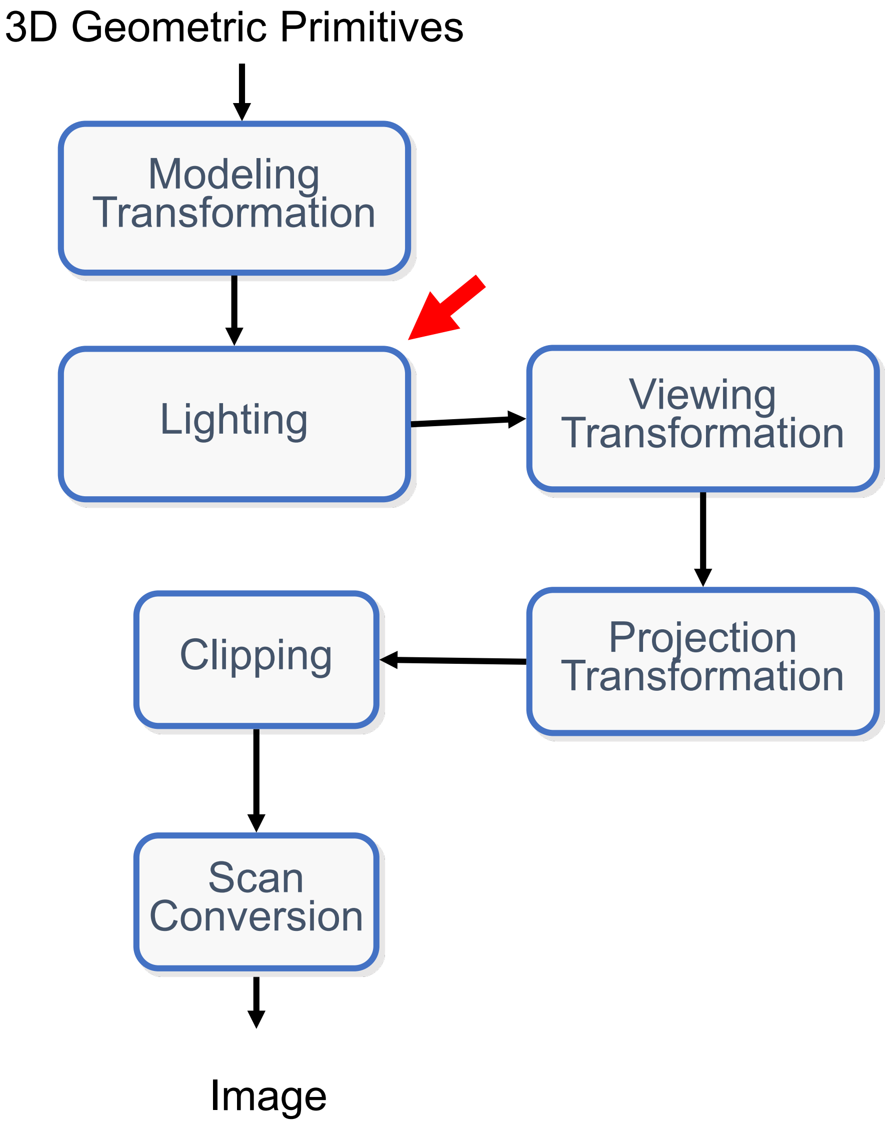


The Graphical Pipeline

- Step 2 – Lighting
  - There are three type of shading approcates:
    - Flat shading – Fast and inaccurate
    - Phong shading – Accurate but slow
    - Gouraud shading – somehwere in the middle in terms of speed and accuracy

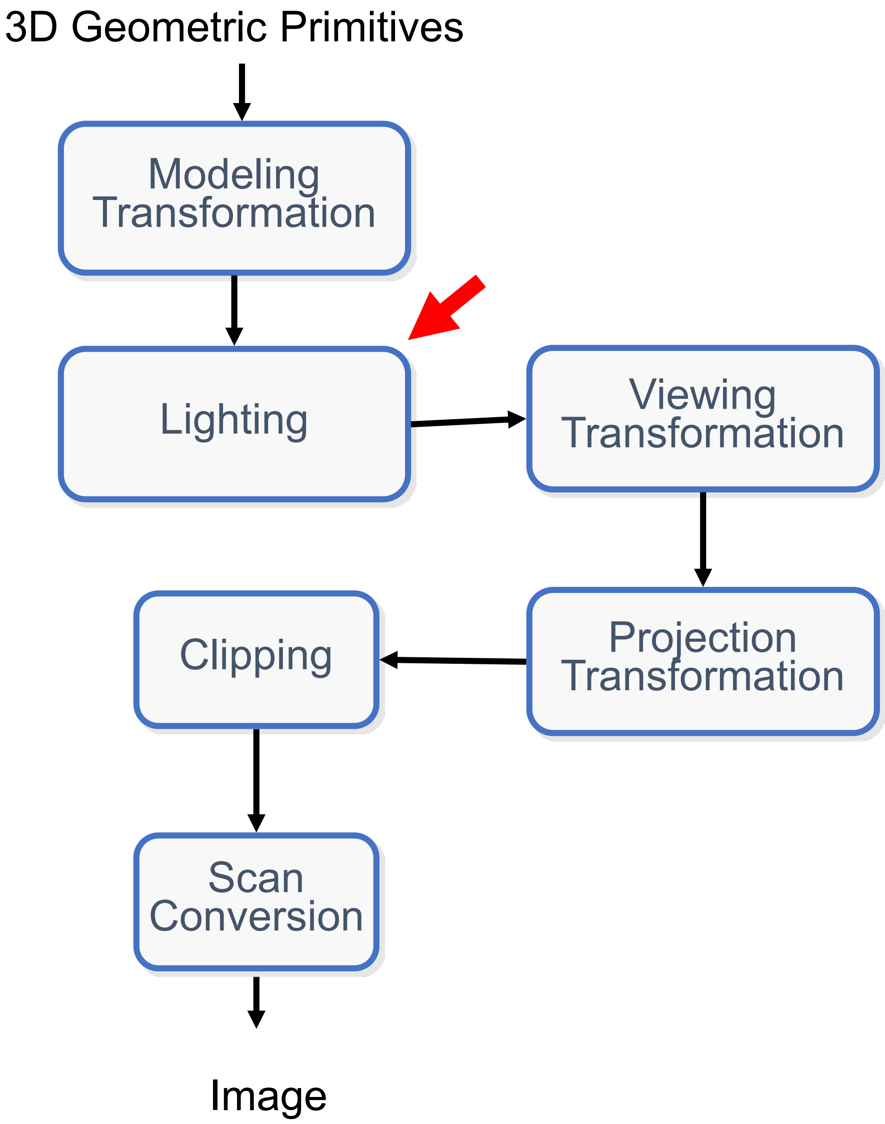


https://adambadke.com/portfolio-single/ray-tracing-3d-renderer/



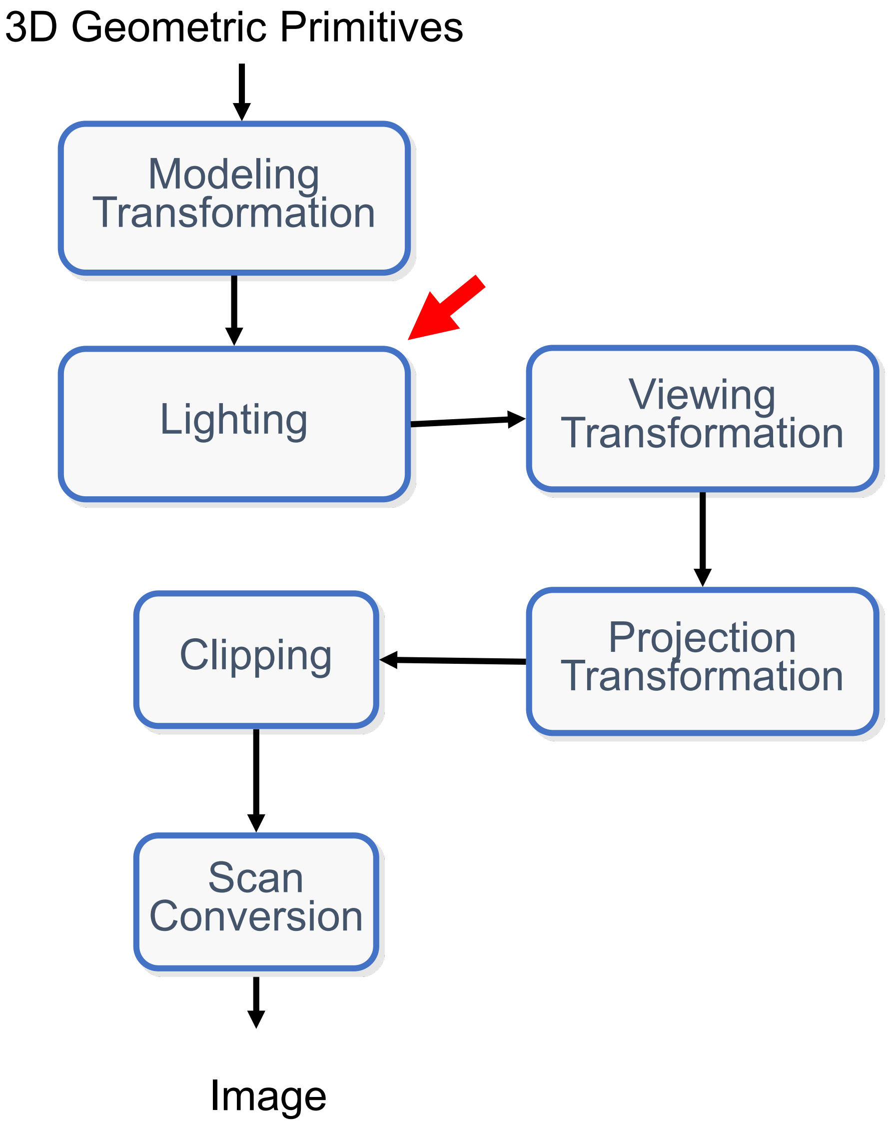
The Graphical Pipeline

- Step 2 – Lighting
  - Flat Shading:
    - Each **"triangle"** is given a normal (the normal can be predefined or calculated using simple geometry).
    - The color of the triangle is determined using some reflection model (e.g Phong reflection model)
    - The color is uniform across each triangle (all points on each triangle have the same color)



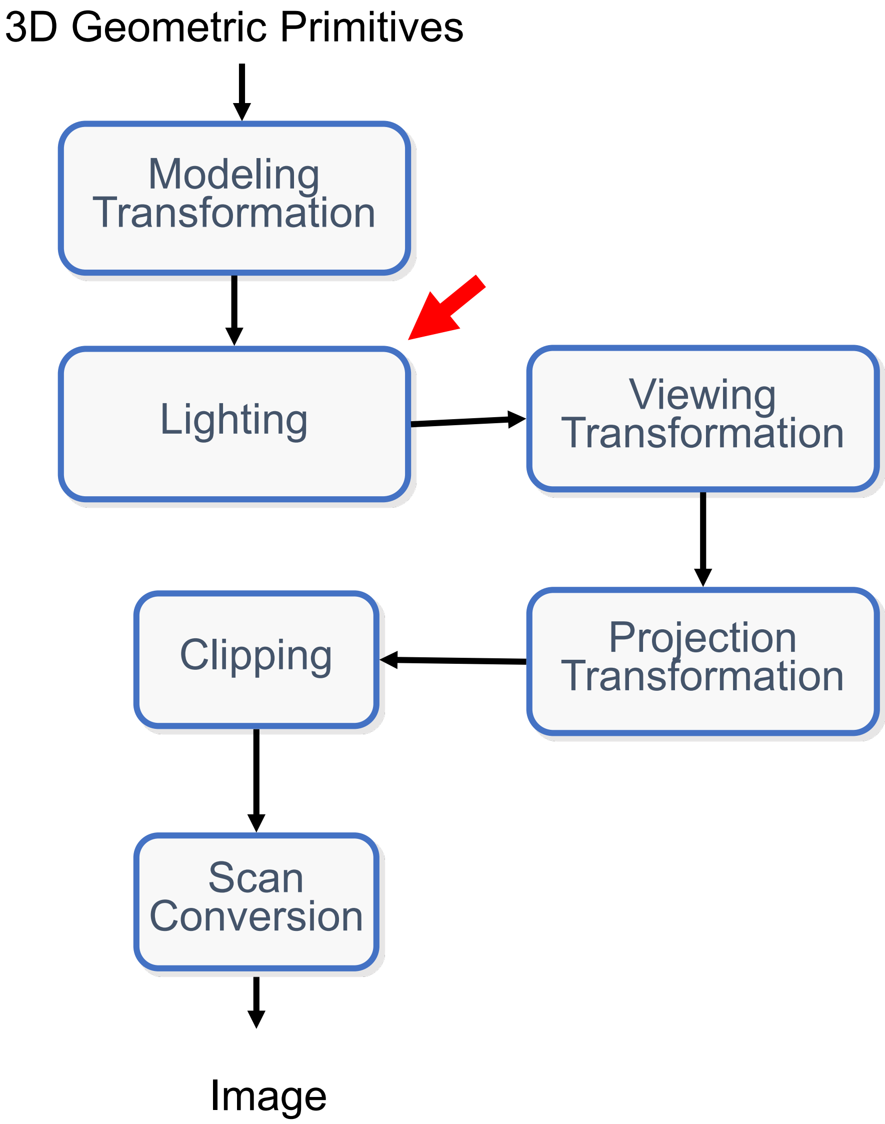
The Graphical Pipeline

- Step 2 – Lighting
  - Gouroaud Shading:
    - Each “triangle **vertex”** is given a normal.
    - The **vertex** color is determined using a reflection model (e.g phong reflection model)
    - The color of a point inside the triangle is an interpolation of the colors of the triangle vertices



The Graphical Pipeline

- Step 2 – Lighting
  - Phong Shading:
    - We define a normal for each triangle vertex
    - The normal of any point point inside the triangle is an interpolation between the triangle vertices' normal vectors.
    - The point color is determined using Phong reflection model with the interpolated normal.

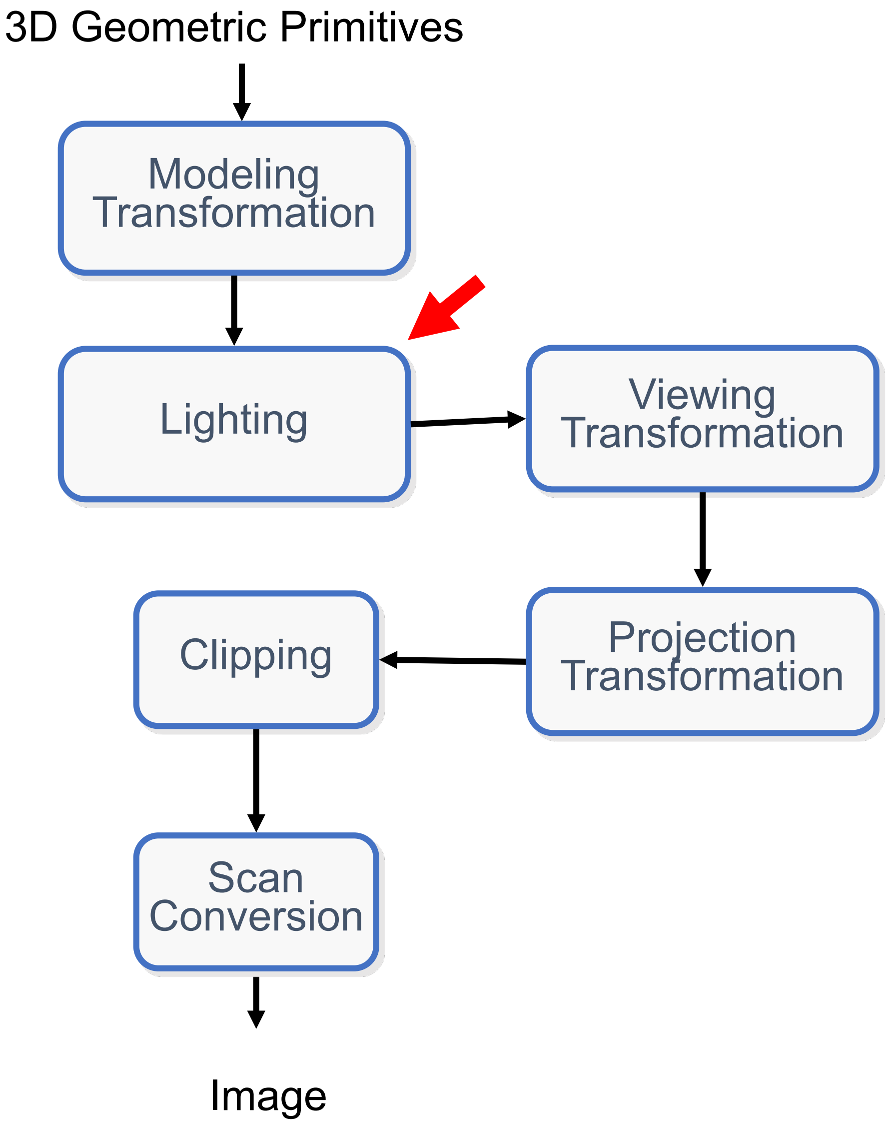


The Graphical Pipeline

- Step 2 – Lighting
  - Flat Shading
  - Gouroud Shading
  - Phong Shading

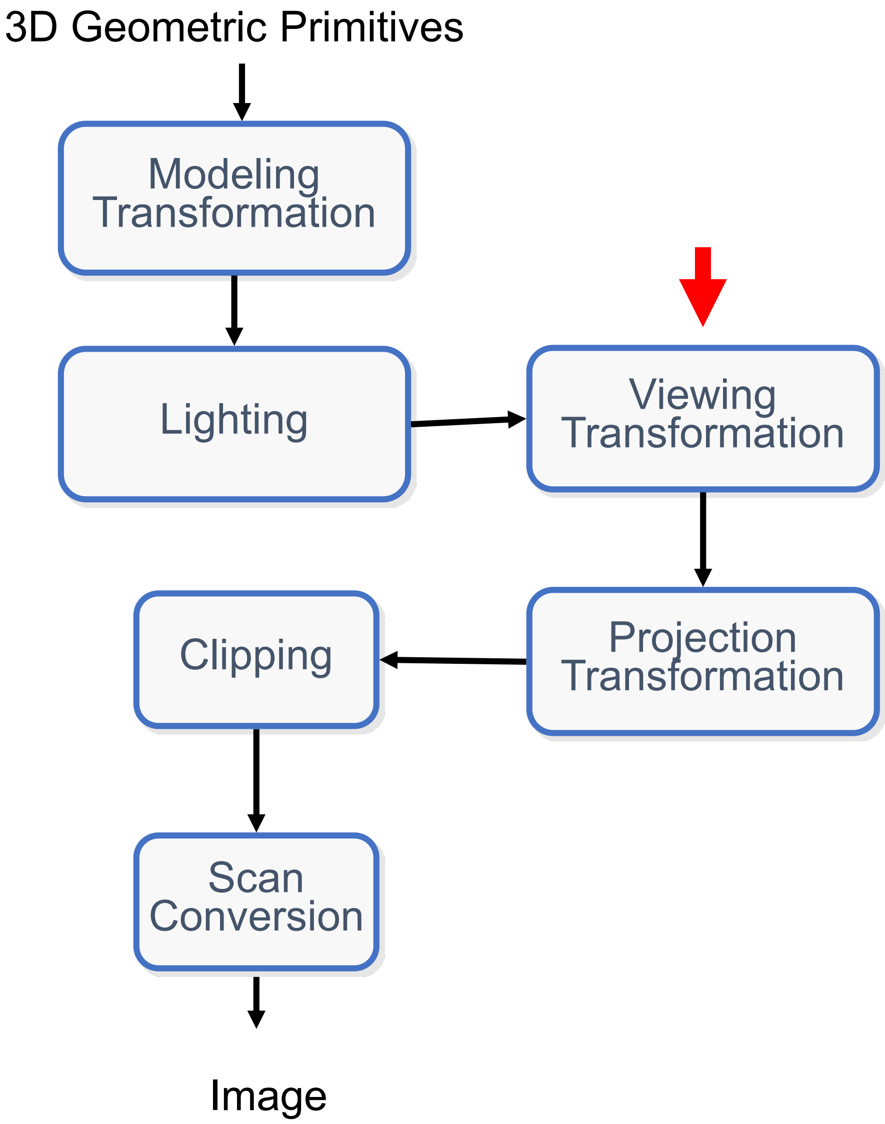
Less expensive

More accurate



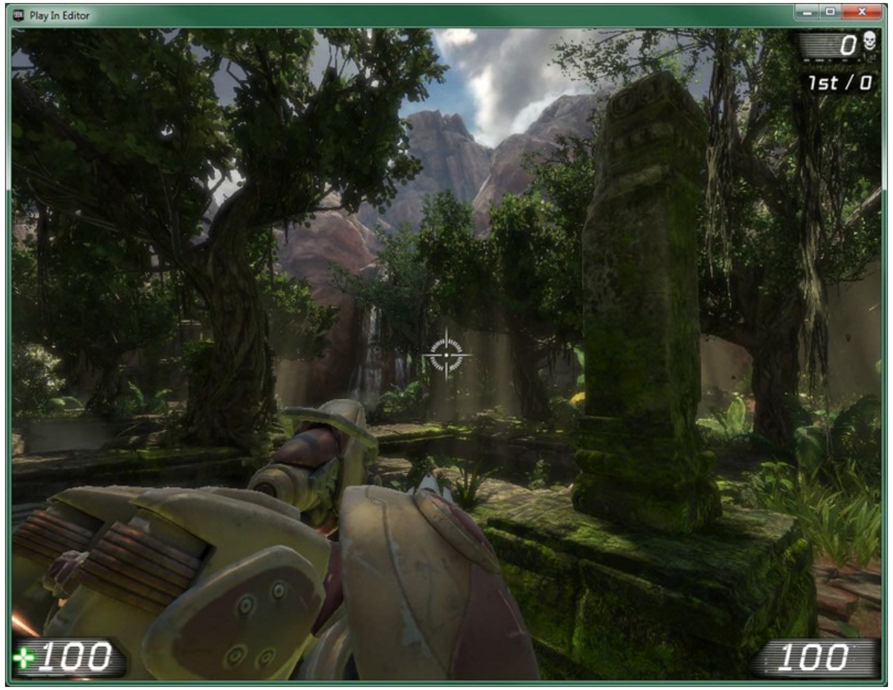
The Graphical Pipeline

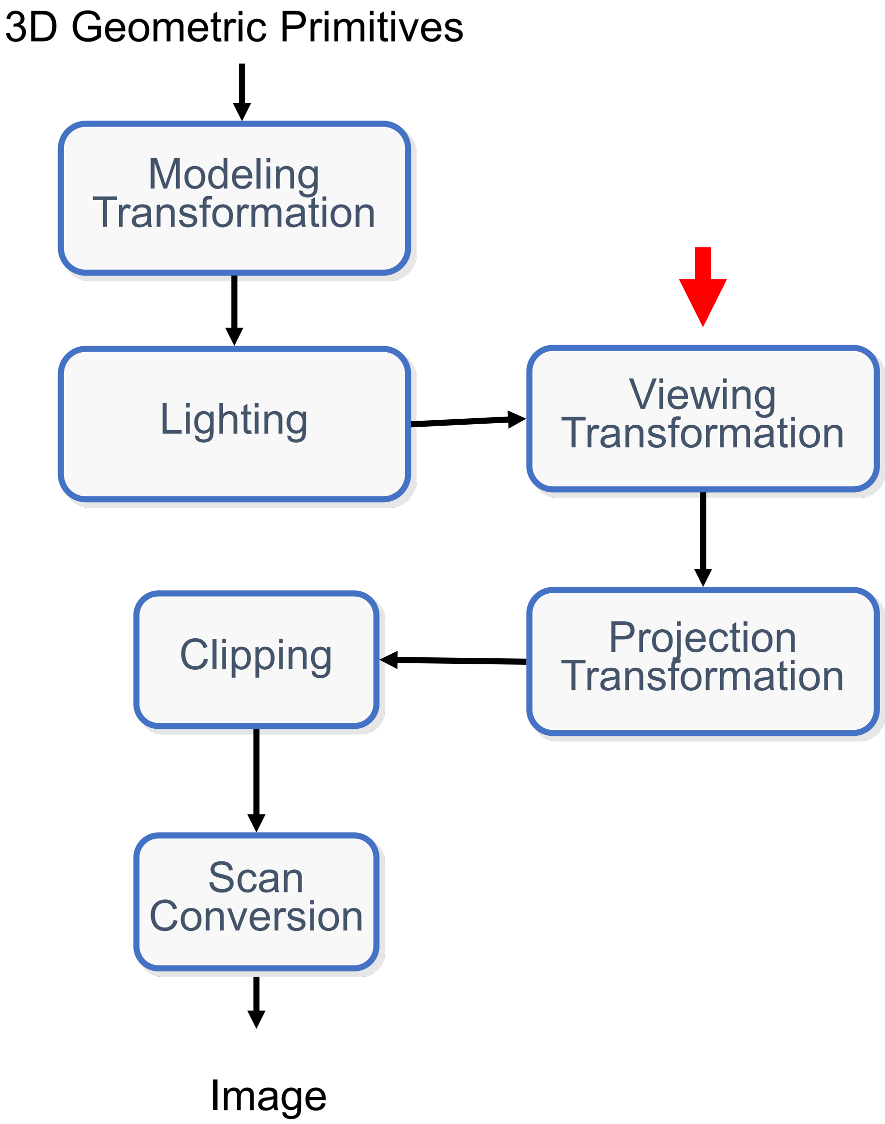
- Step 3 – Viewing Transformation
  - The final image is highly dependent on the camera location and view direction:
    - What we see is what we render



The Graphical Pipeline

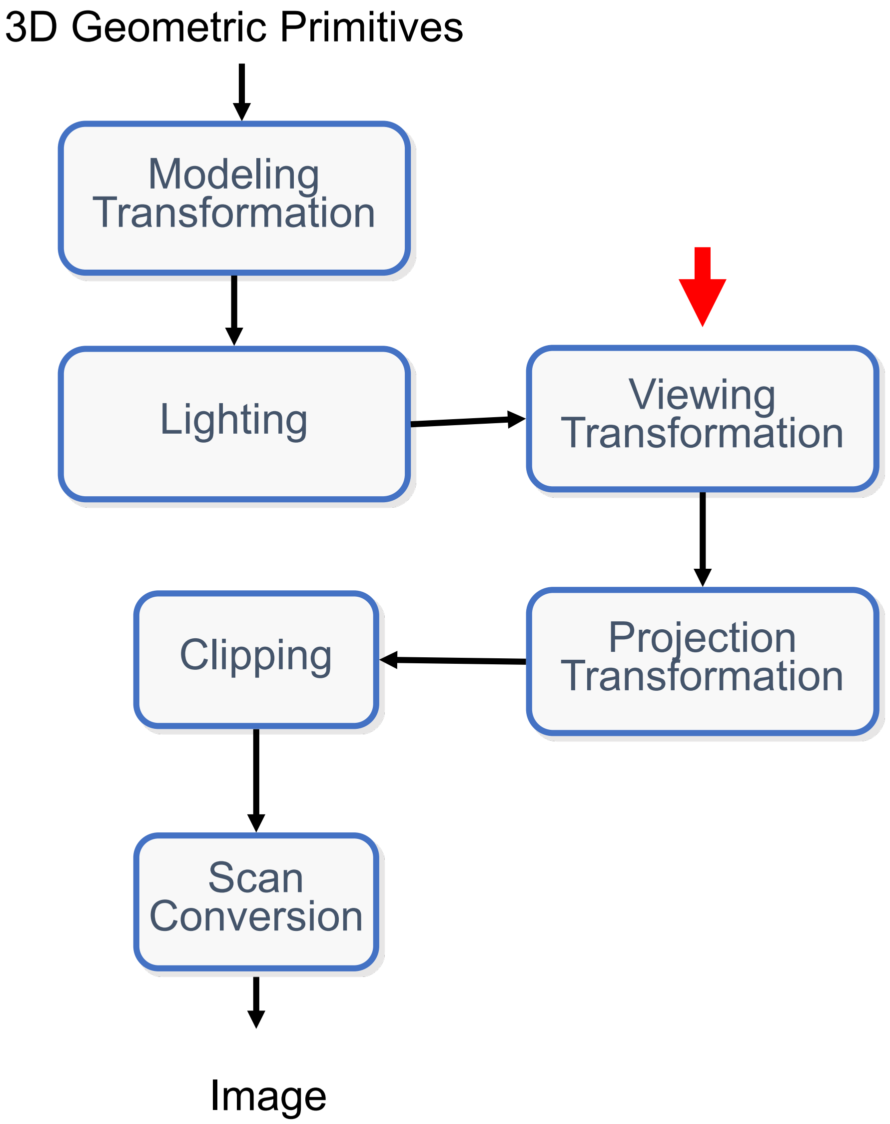
- Step 3 – Viewing Transformation
  - The final image is highly dependent on the camera location and view direction:
    - What we see is what we render





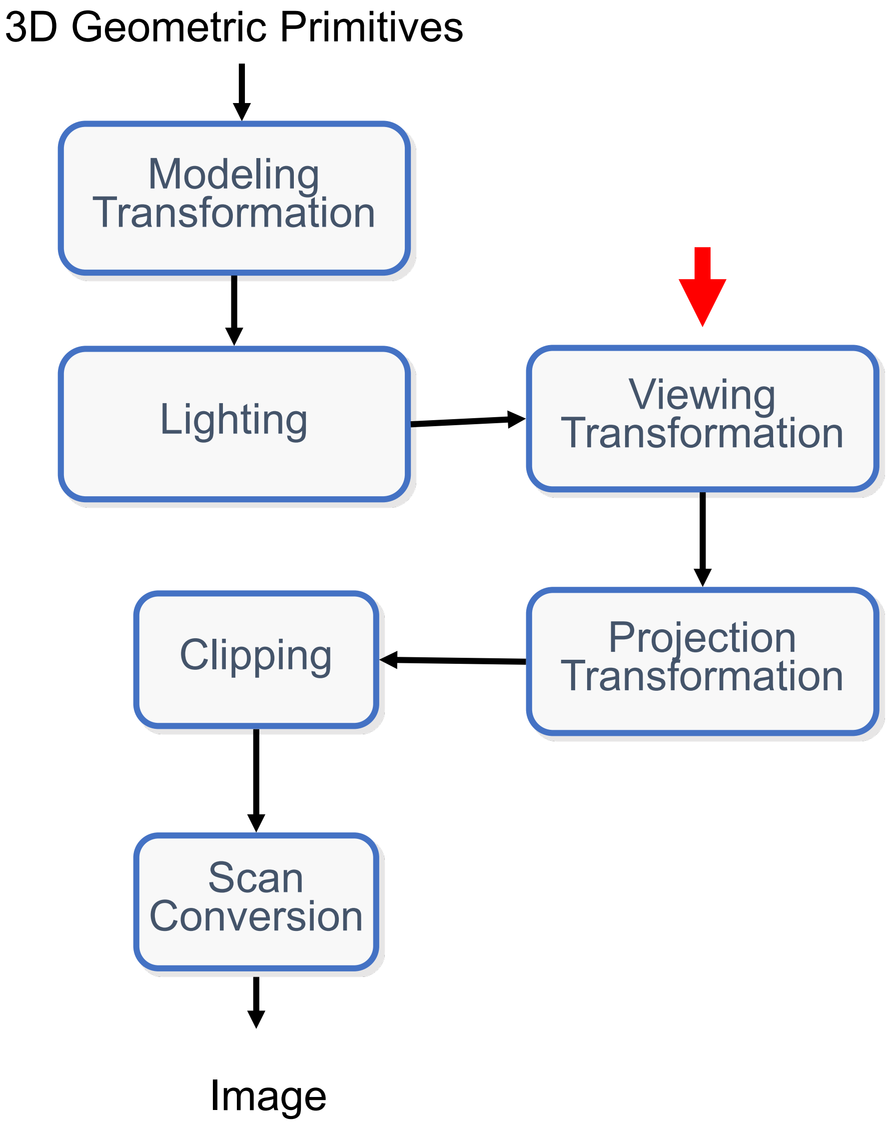
The Graphical Pipeline

- Step 3 – Viewing Transformation
  - We therefore need to define the camera parameters:
    - Location
    - View direction
    - View volume (more later)



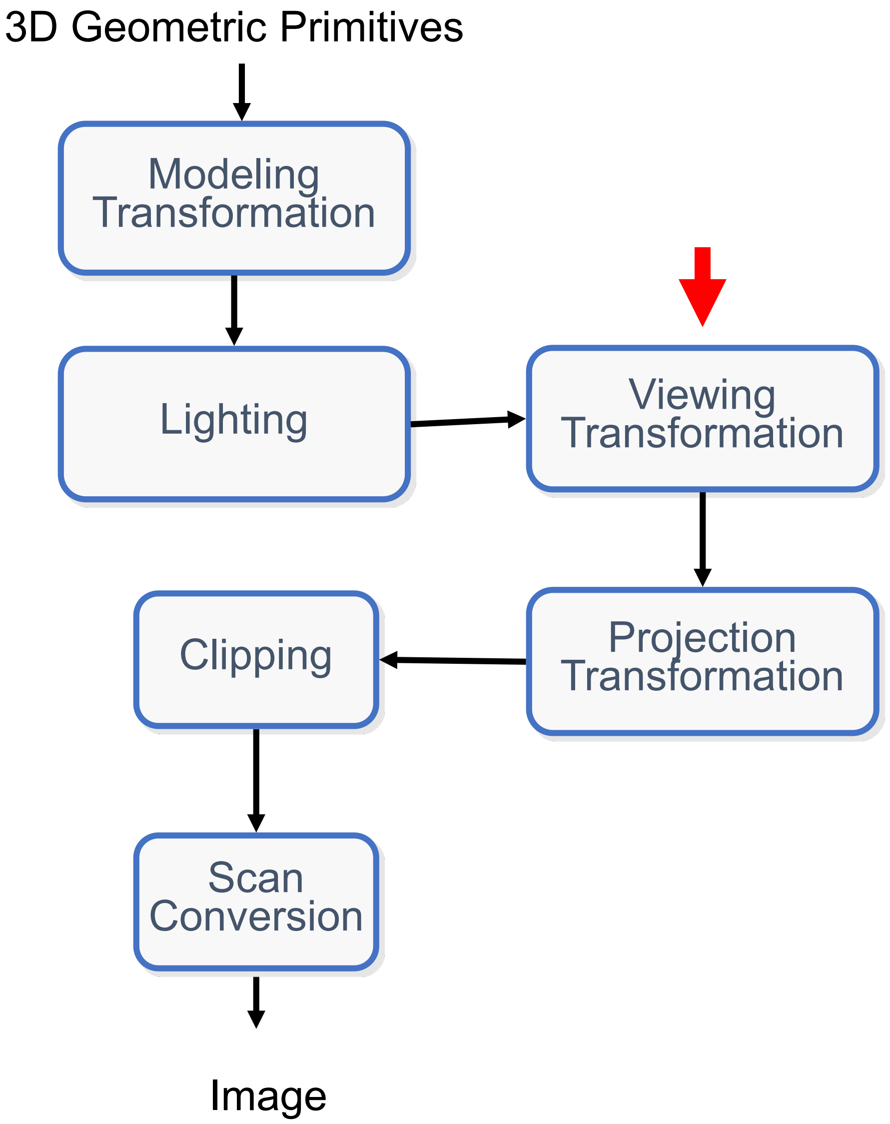
The Graphical Pipeline

- Step 3 – Viewing Transformation
  - In the graphical pipeline this is done slightly different
  - Instead of defining the camera parameters, we always assume the camera is in the origin and is looking in the negative z-direction
  - This is called the canonical camera parameters
  - This assumption is needed for efficient implementation of different algorhtms



The Graphical Pipeline

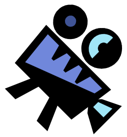
- Step 3 – Viewing Transformation
  - But how can we still support different camera views if the camera is always in the origin and looking in –z direction ?
  - Simple: transform the **whole 3D scene to the camera coordinate system**
  - Therefore, each vertex will be transformed by (at least) two transformation
    - The Model Transformation matrix (local -> world)
    - The Viewing Transformation Matrix (world -> camera)

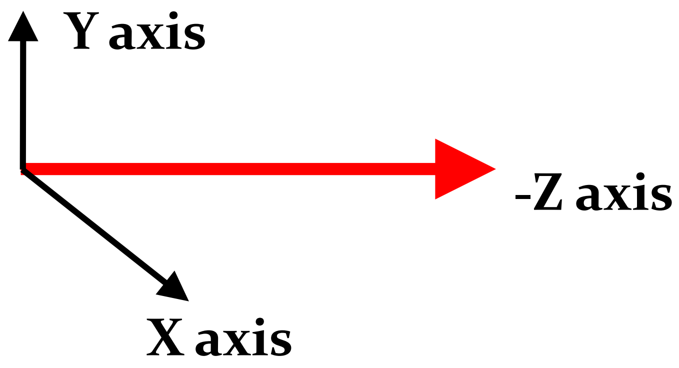


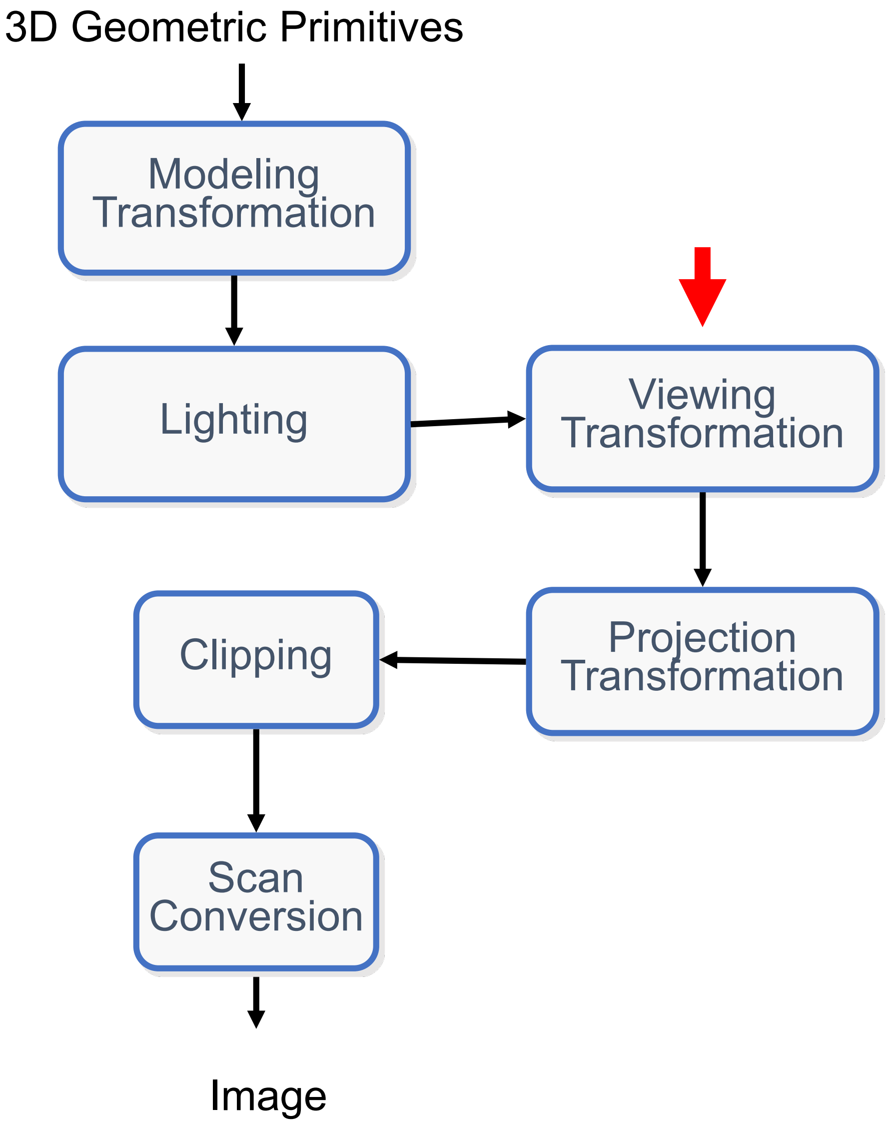
The Graphical Pipeline

- Step 3 – Viewing Transformation
  - Example:
    - We want the camera to be at (0,8,10)
    - And looking in the $\left(0\,,\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)$ direction:

(0,8,10)

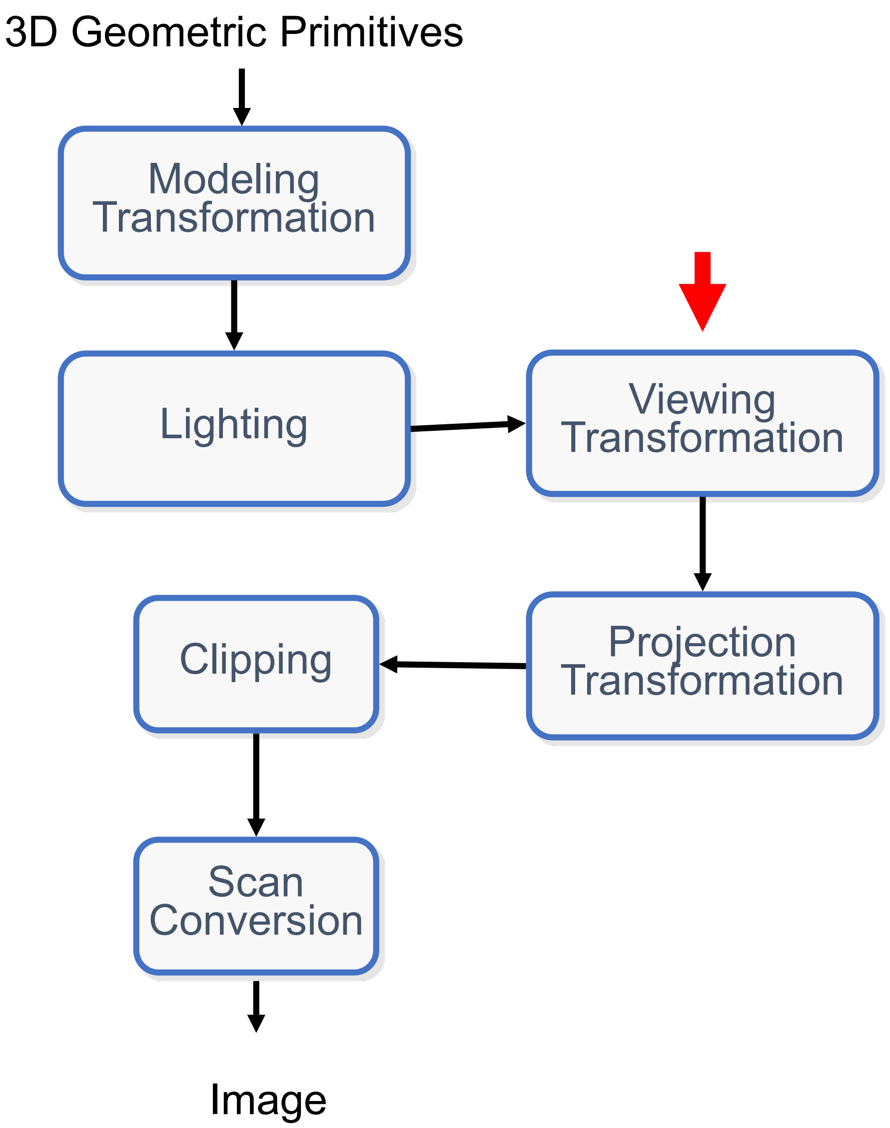






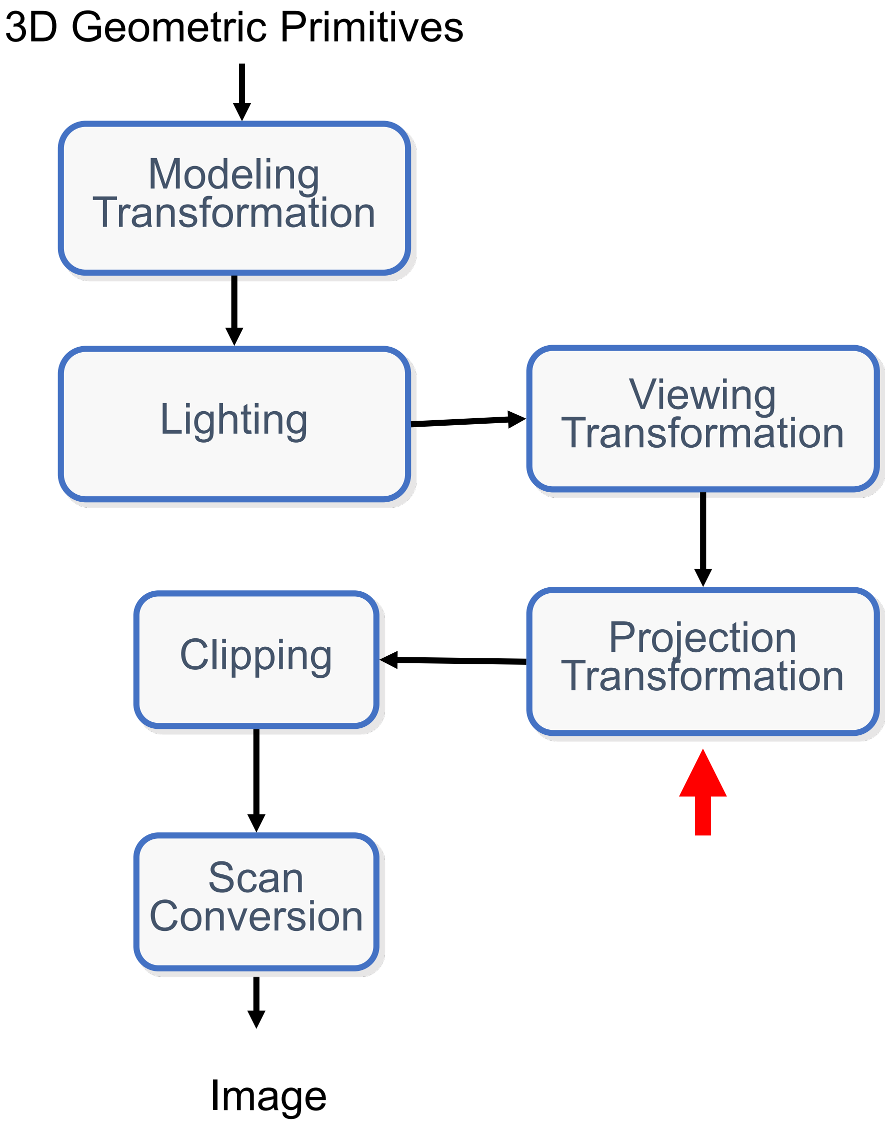
The Graphical Pipeline

- Step 3 – Viewing Transformation
  - Example:
    - First: translate every vertex in the scene by (0,-8,-10)
    - Next: Rotate the whole scene by 45 degrees CCW in the x-direction
    - This can be done using one matrix (e.g. multiplication of the translation and rotation matrix)



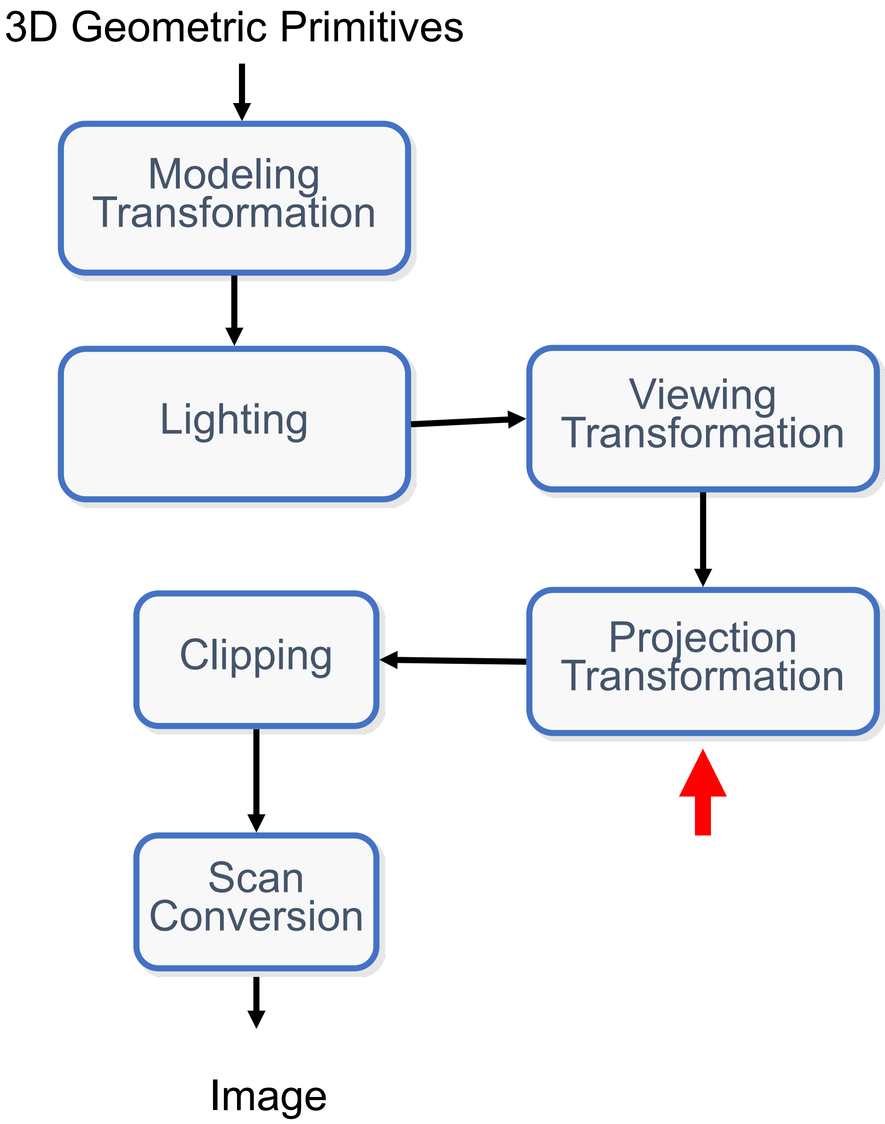
The Graphical Pipeline

- Step 4 – Projection Transformation
  - Until this stage – everything we work with is in 3D space
  - Images on the other hand are in 2D space
  - But how can we move from 3D to 2D space?
    - Use projection transformations:
    - Perspective and Orthographic transformation ?



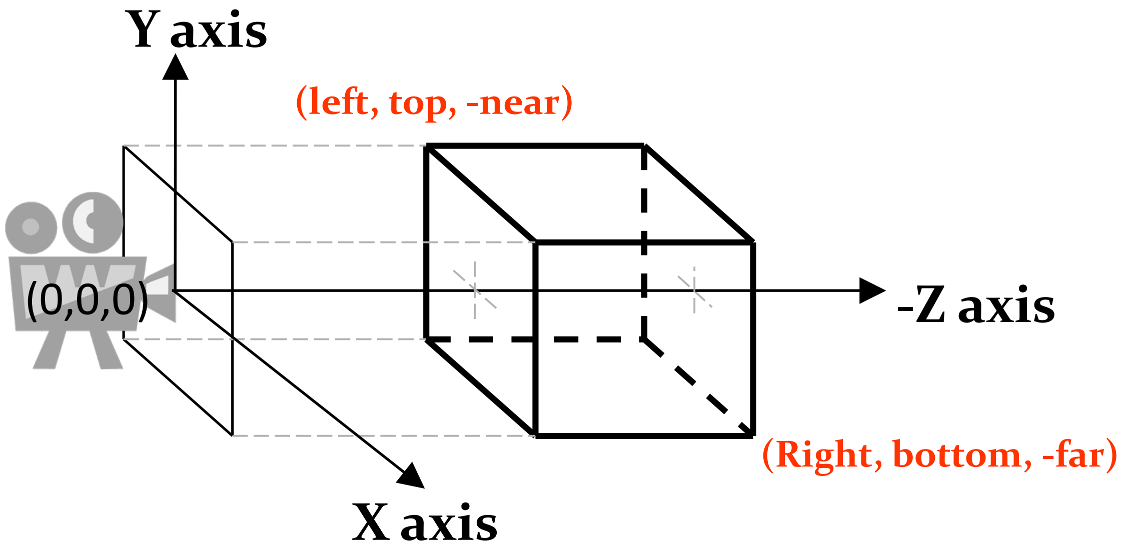
The Graphical Pipeline

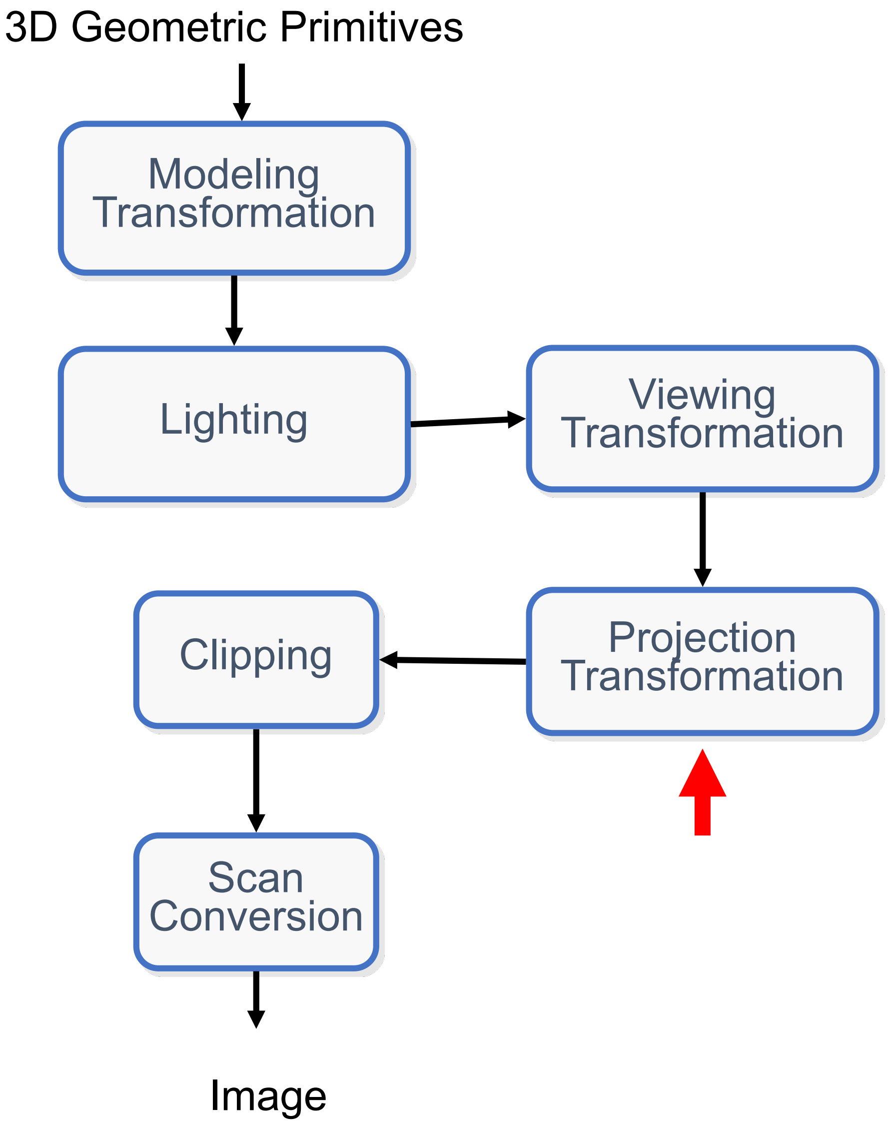
- Step 4 – Projection Transformation
    - Remember the canonical camera coordinates?
    - It simplifies the projection transformation matrix



The Graphical Pipeline

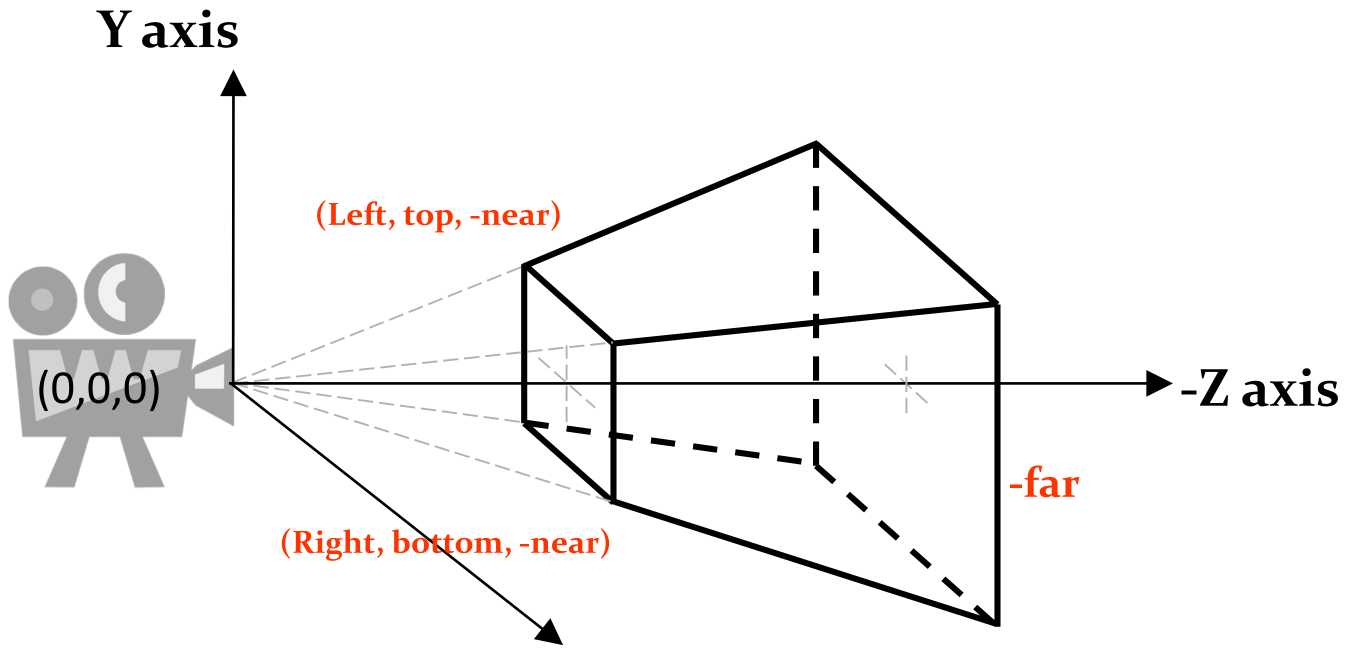
- Step 4 – Projection Transformation
  - Orthographic projection
    - Everything inside the “view volume” will be projected onto the XY plane
    - Everything else will be clipped

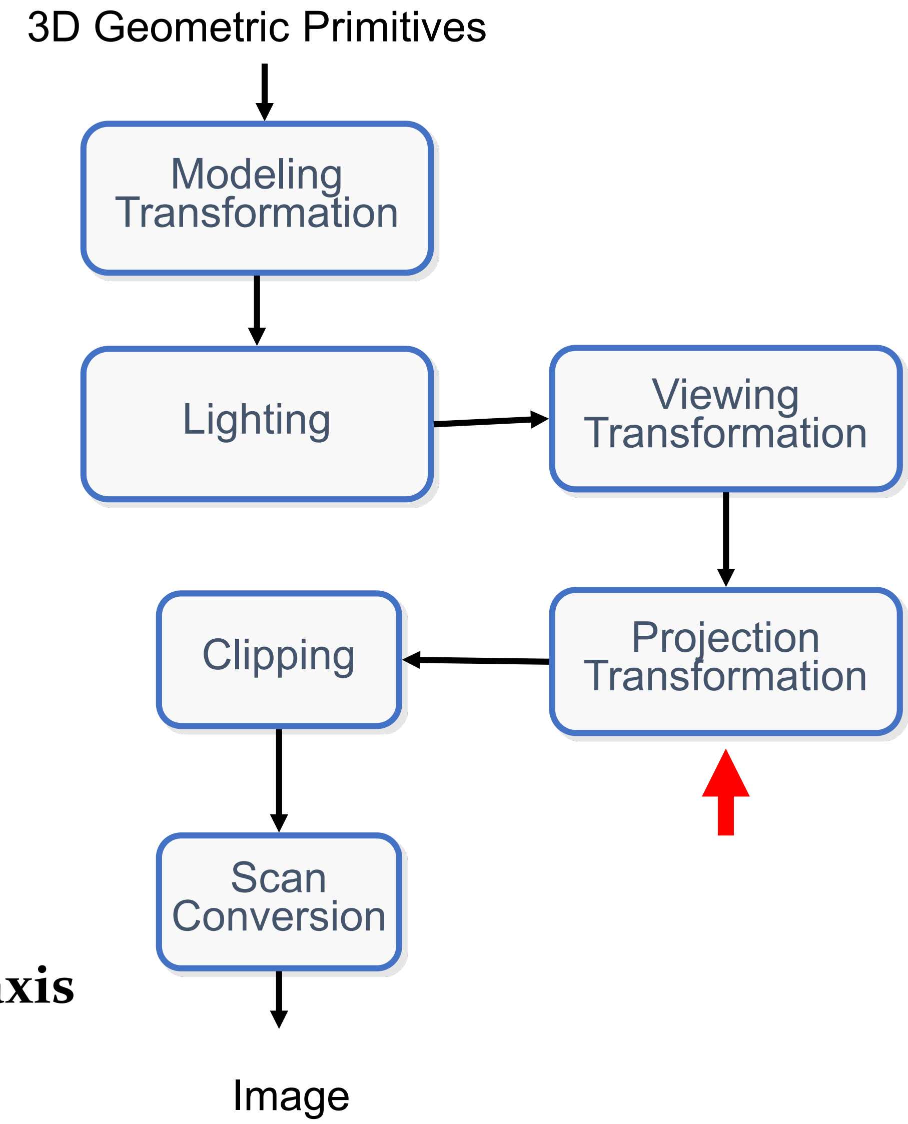




The Graphical Pipeline

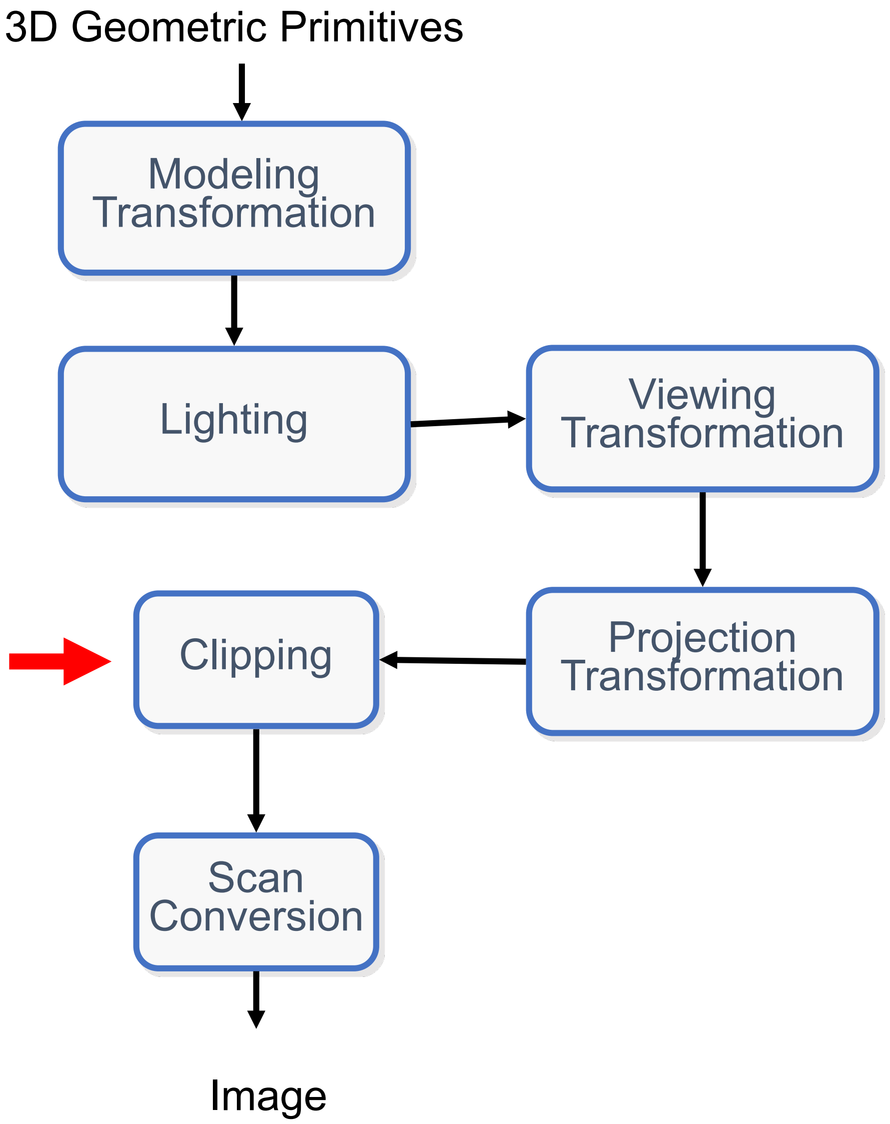
- Step 4 – Projection Transformation
  - Perspective projection
    - Everything inside the "view frustum volume" will be projected onto the near frustum plane
    - Everything else will be clipped





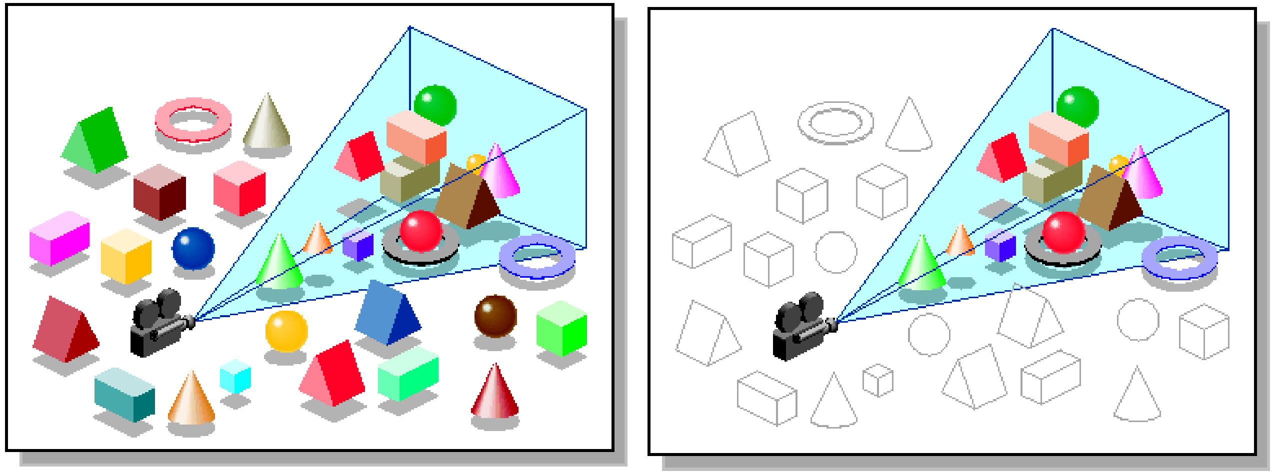
The Graphical Pipeline

- Step 5 – Clipping
  - A 3D scene is composed of many objects
    - Each object is composed of thousands of triangles (even more)
  - But not everything is visible – why bother and render everything if not everything is visible ?
  - Therefore we clip-out objects outside the view-frustum:
    - This is important process which is important for real-time rendering
    - Done for efficiency purposes

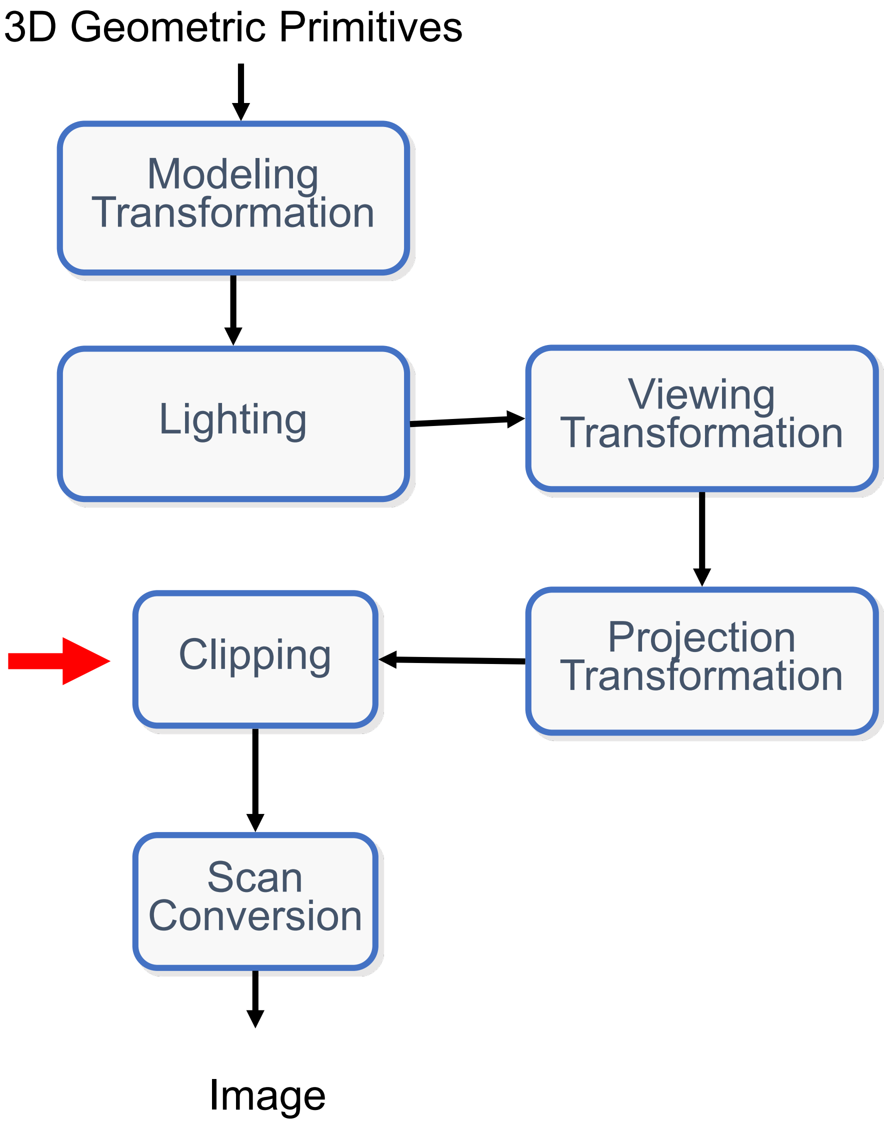


The Graphical Pipeline

- Step 5 – Clipping
  - Faded objects are not rendered:

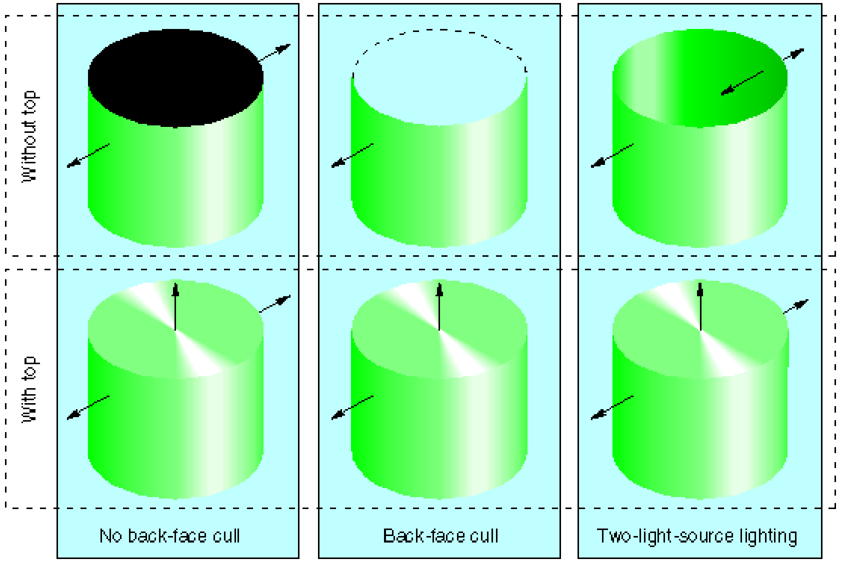


https://techpubs.jurassic.nl/manuals/0640/developer/Optimizer_PG/sgi_html/ch05.html

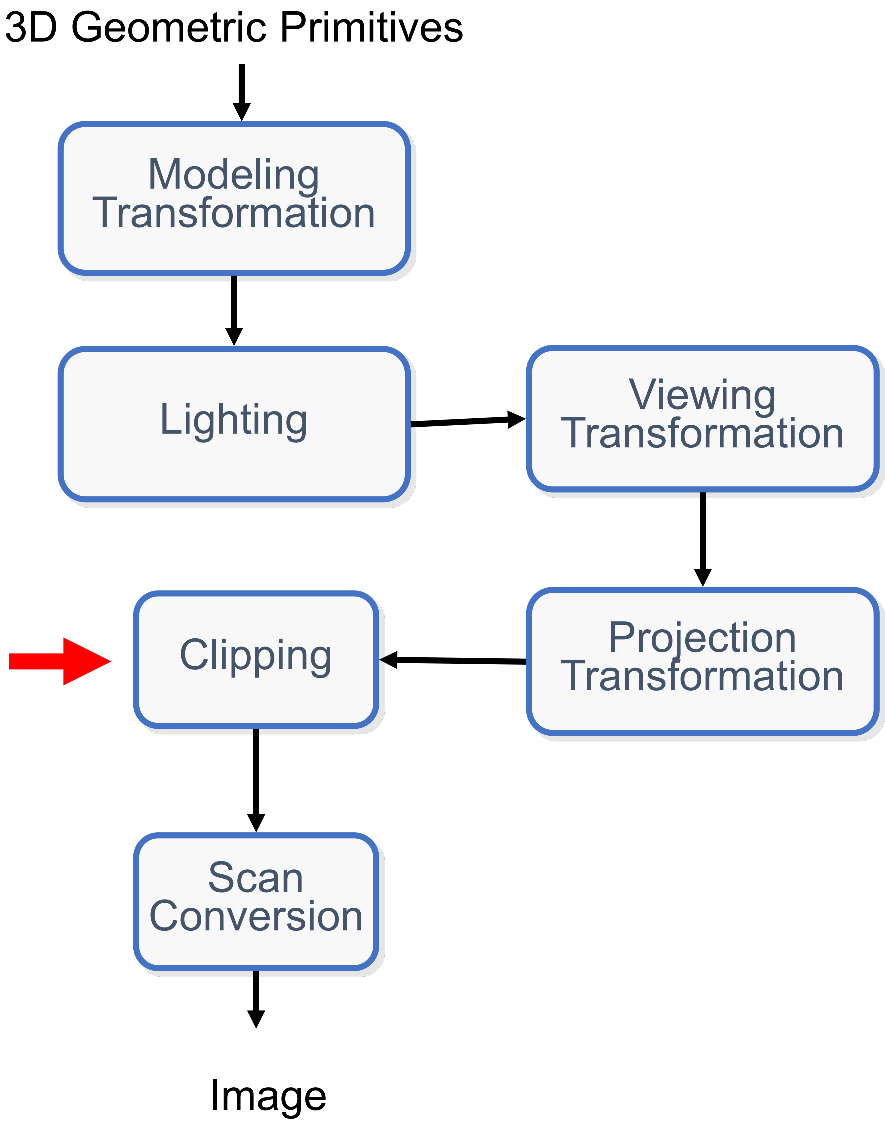


The Graphical Pipeline

- Step 5 – Clipping
  - Typically, triangles should not be rendered when their front sides do not face the viewpoint.
  - Such pieces of a surface are called *back faces.*

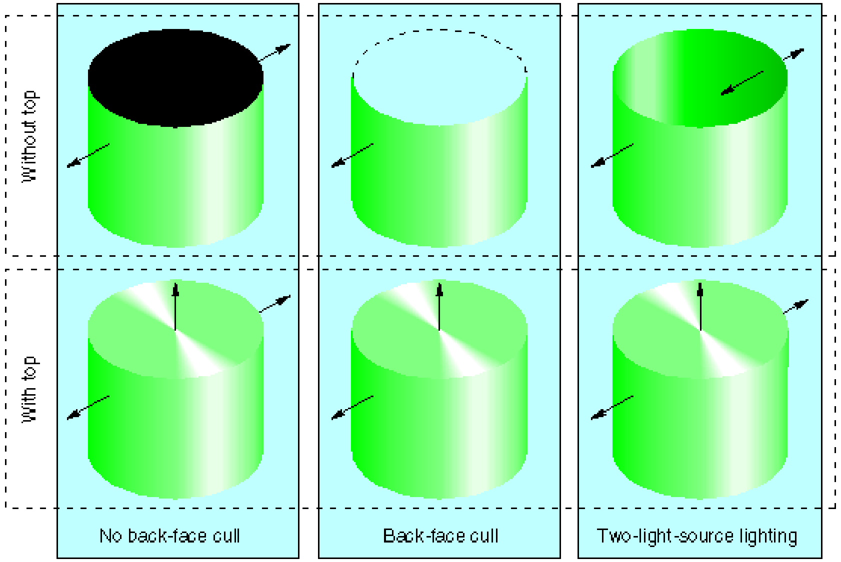


https://techpubs.jurassic.nl/manuals/0640/developer/Optimizer_PG/sgi_html/ch05.html

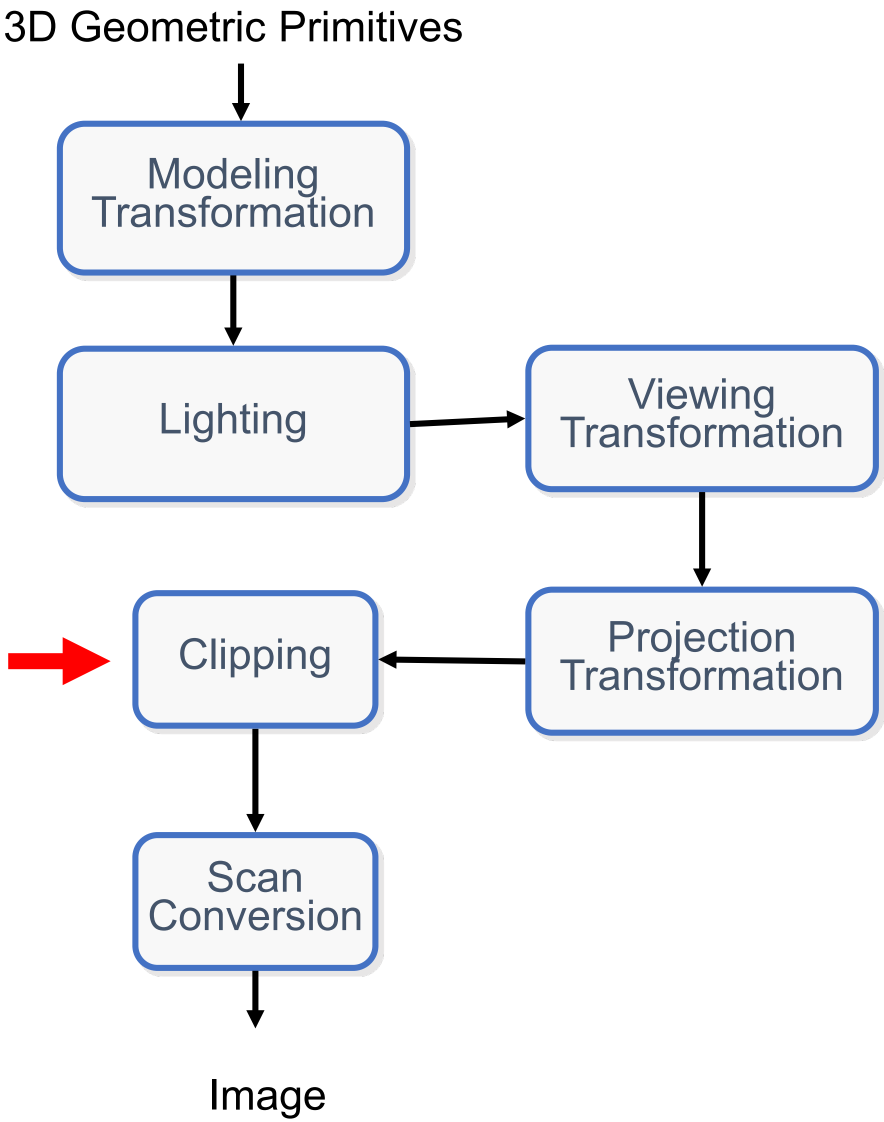


The Graphical Pipeline

- Step 5 – Clipping
  - **Back-face culling** keeps these triangles from being rendered (rasterized), thus saving on pixel fill time (later on rasterization).

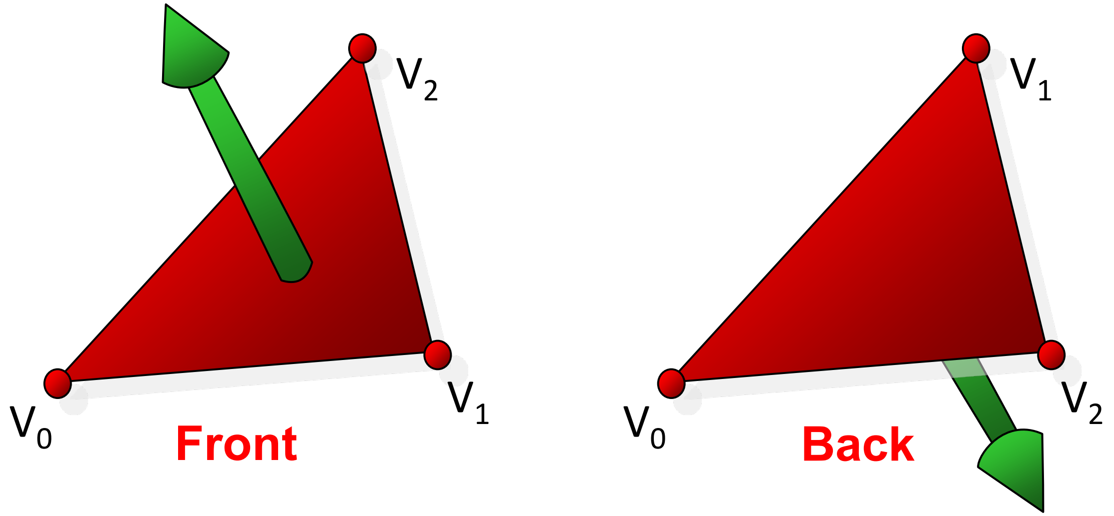


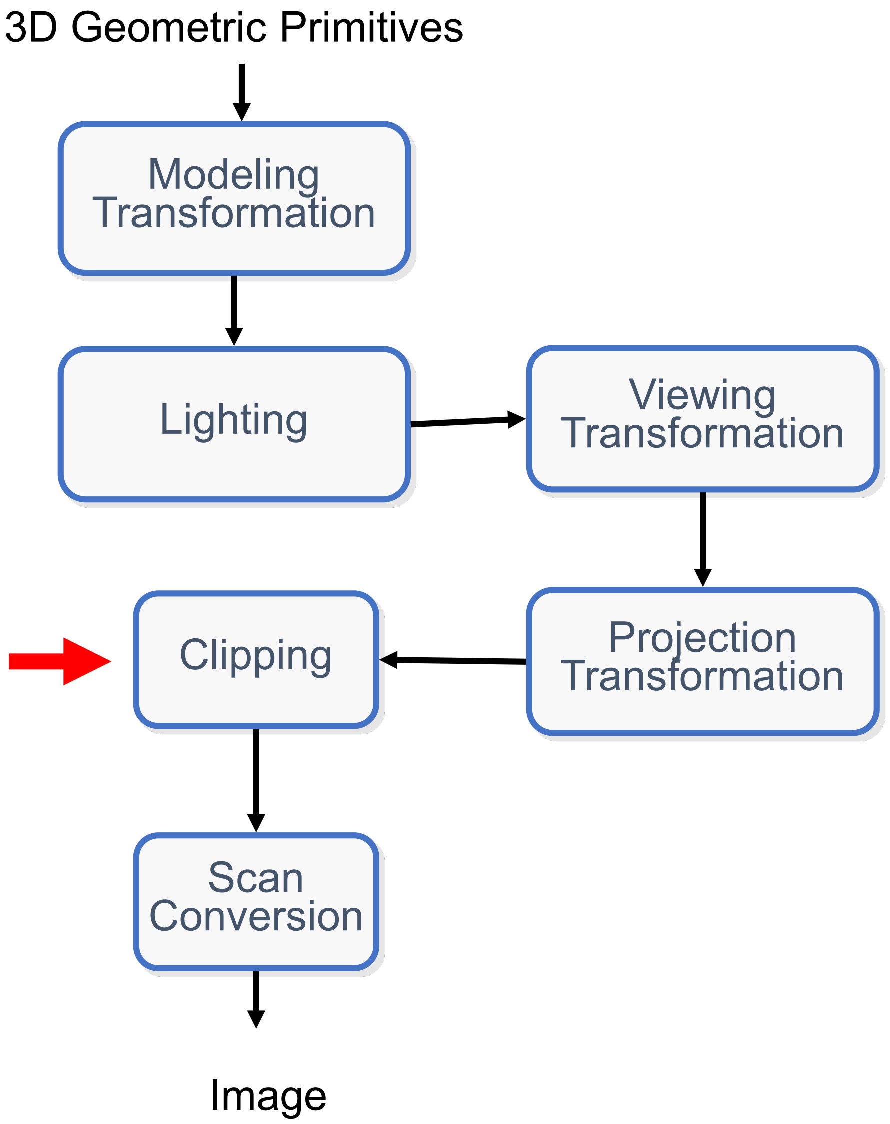
https://techpubs.jurassic.nl/manuals/0640/developer/Optimizer_PG/sgi_html/ch05.html



The Graphical Pipeline

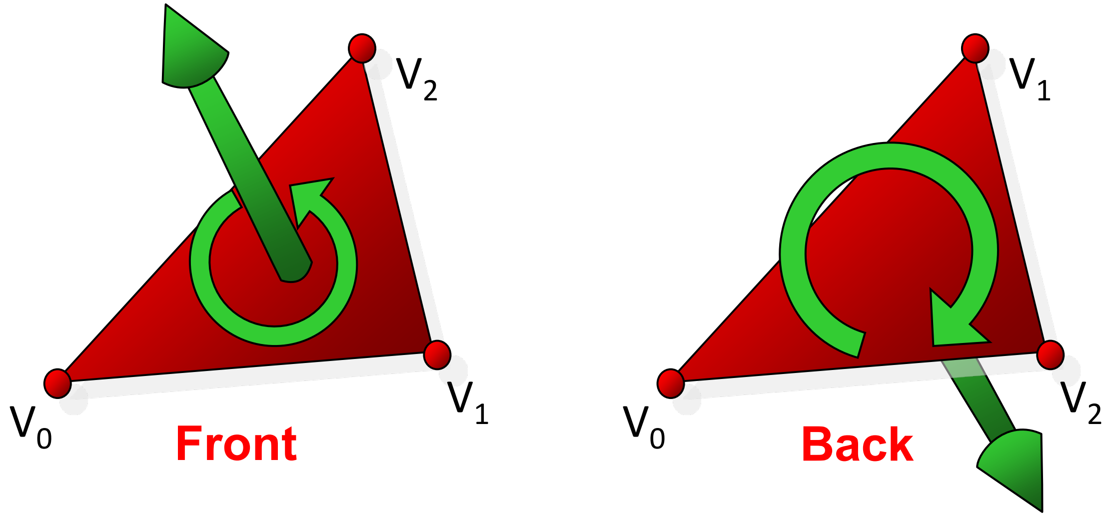
- Step 5 – Clipping
  - Face orientation is determined by the surface normal.
    - If the surface normal points towards the camera then the visible surface is the front face.
    - otherwise the backface

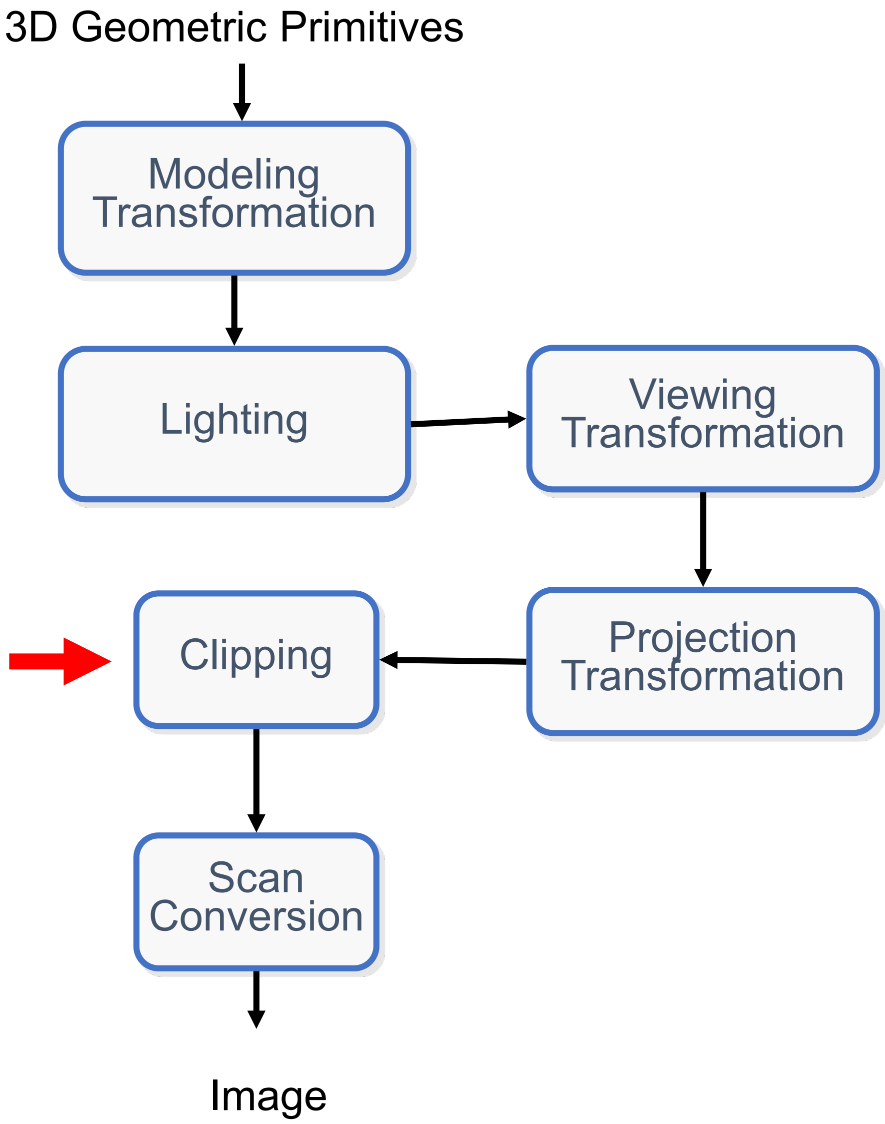




The Graphical Pipeline

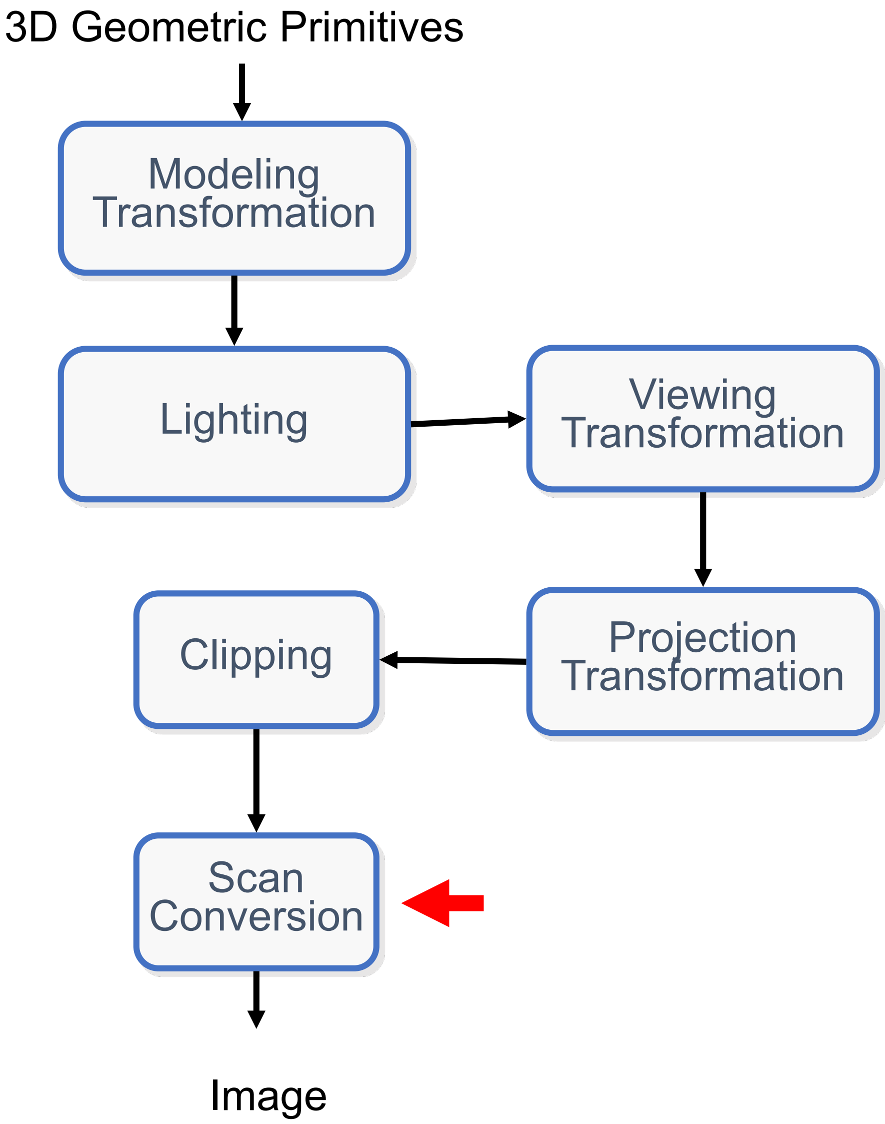
- Step 5 – Clipping
  - This normal is not defined explicitly, but deduced by the order in which the vertices are processed in the pipeline (Counter-ClockWise or Clock-Wise)
  - Use right-hand rule to determine the normal direction
  - Note below, we use the vertex index to represent the order in which the vertices are processed in the pipeline





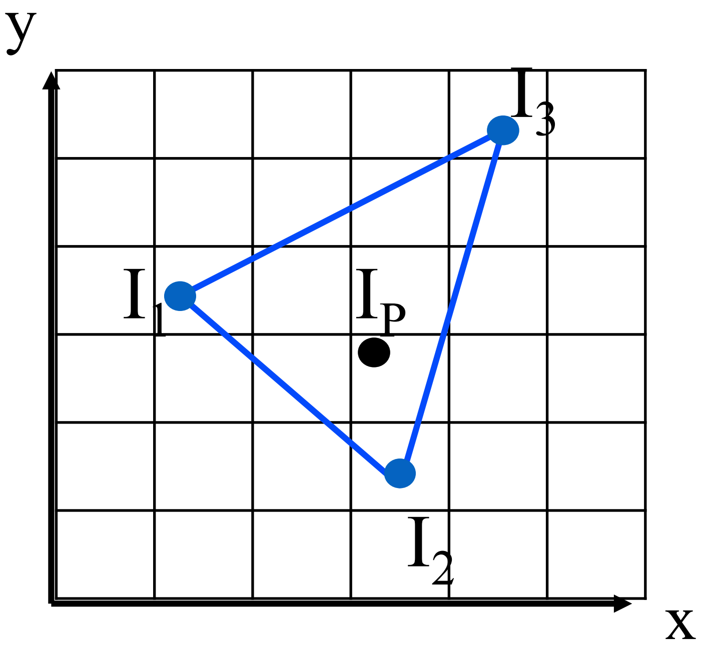
The Graphical Pipeline

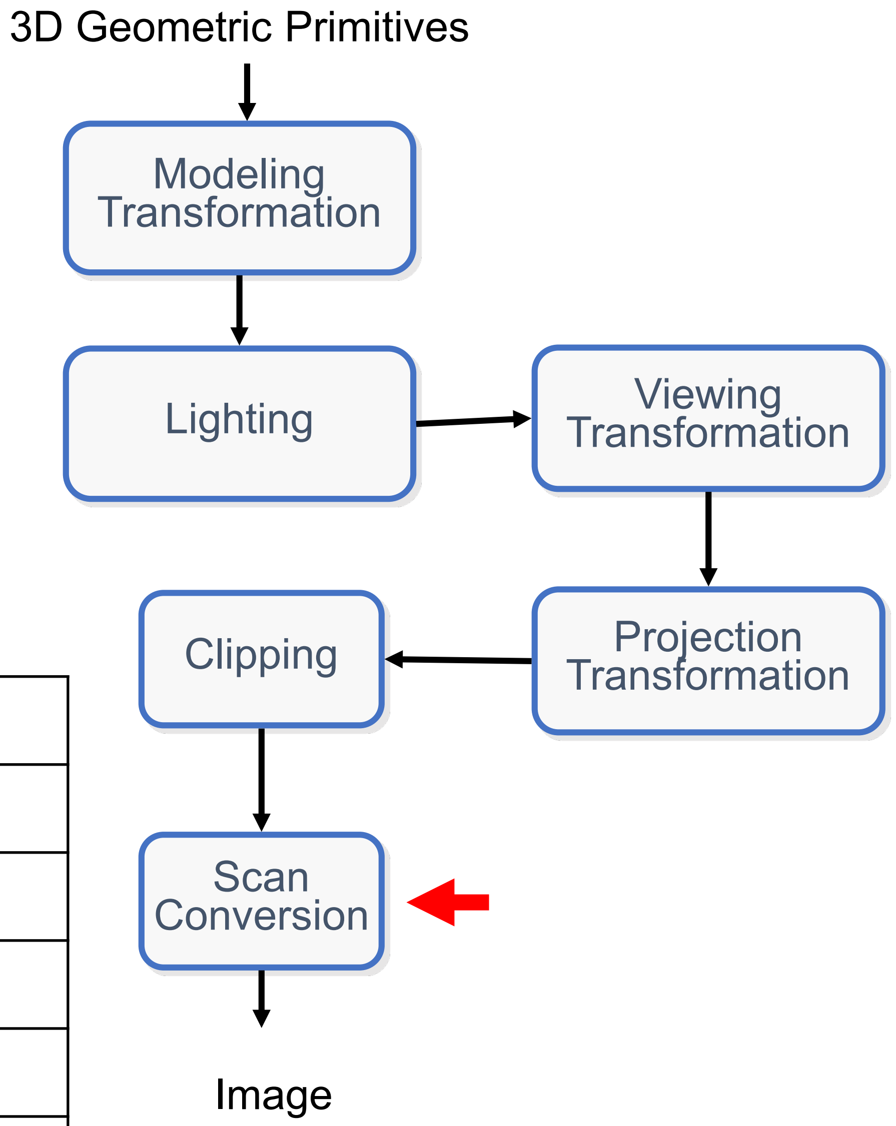
- Step 6 – Scan Conversion
  - aka Rasterization
  - Images are represented in Discrete 2D space:
    - i.e., a grid of pixels
  - Until this stage – everything was done in continous 2D stage.
  - In this stage we convert the vertex information output by the geometry pipeline into pixel information needed by the video display



The Graphical Pipeline

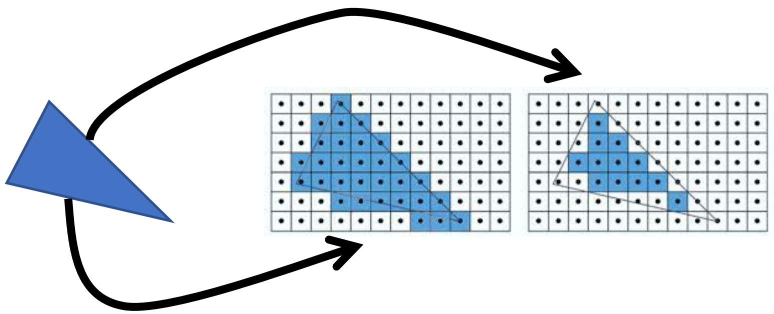
- Step 6 – Scan Conversion
  - In this stage we also determine the fill colors of each triangle
    - Remember the lighting stage ?
    - Everything we calculate in stage 2 will be used to determine the fill color:

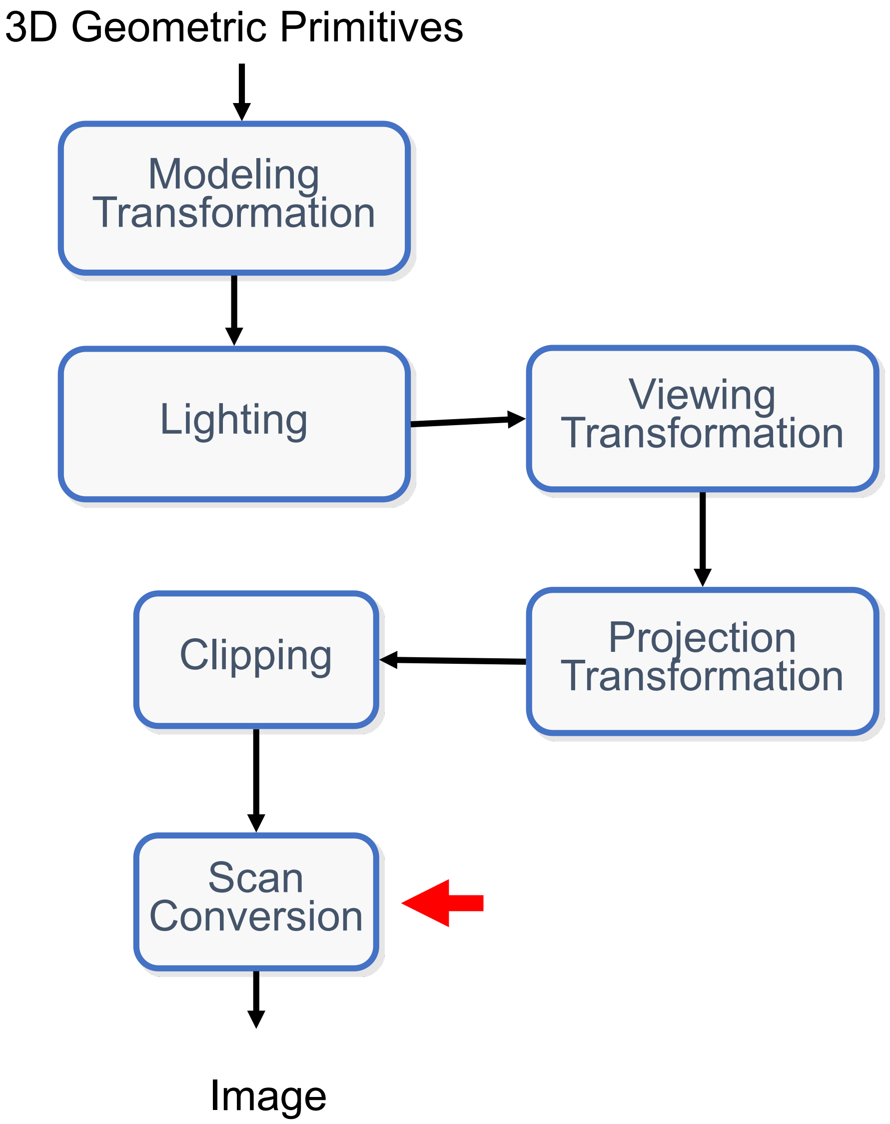




The Graphical Pipeline

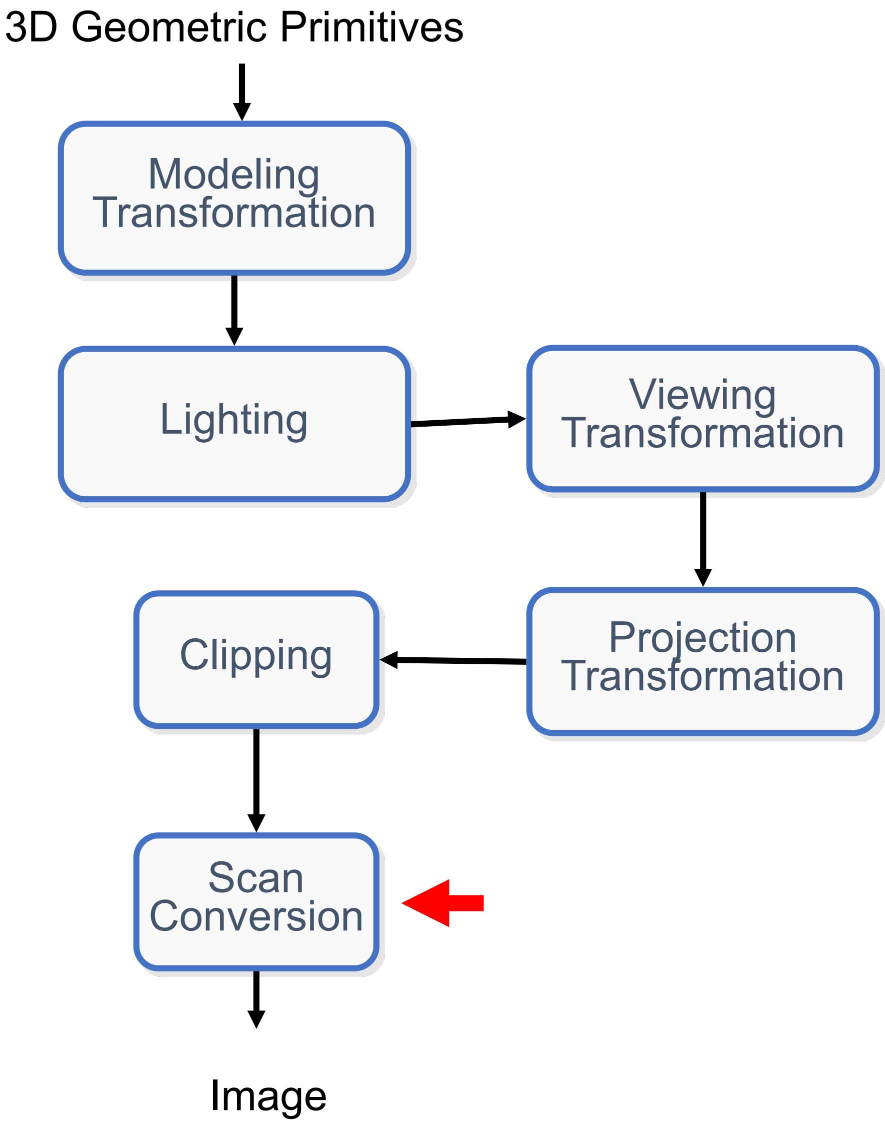
- Step 6 – Scan Conversion
  - Below there are two options for the same triangle rasterization
  - Aliasing: distortion artifacts produced when representing a high-resolution signal at a lower resolution.
  - In this stage we handle aliasing artifacts using anti-aliasing algorithms





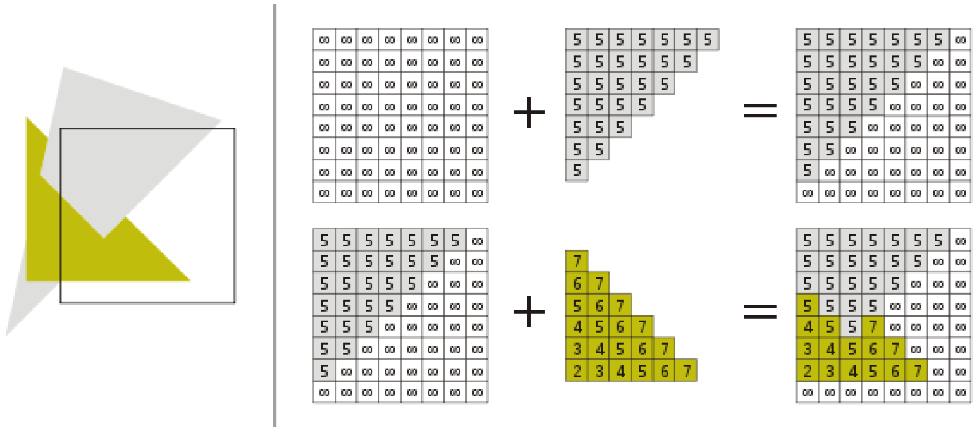
The Graphical Pipeline

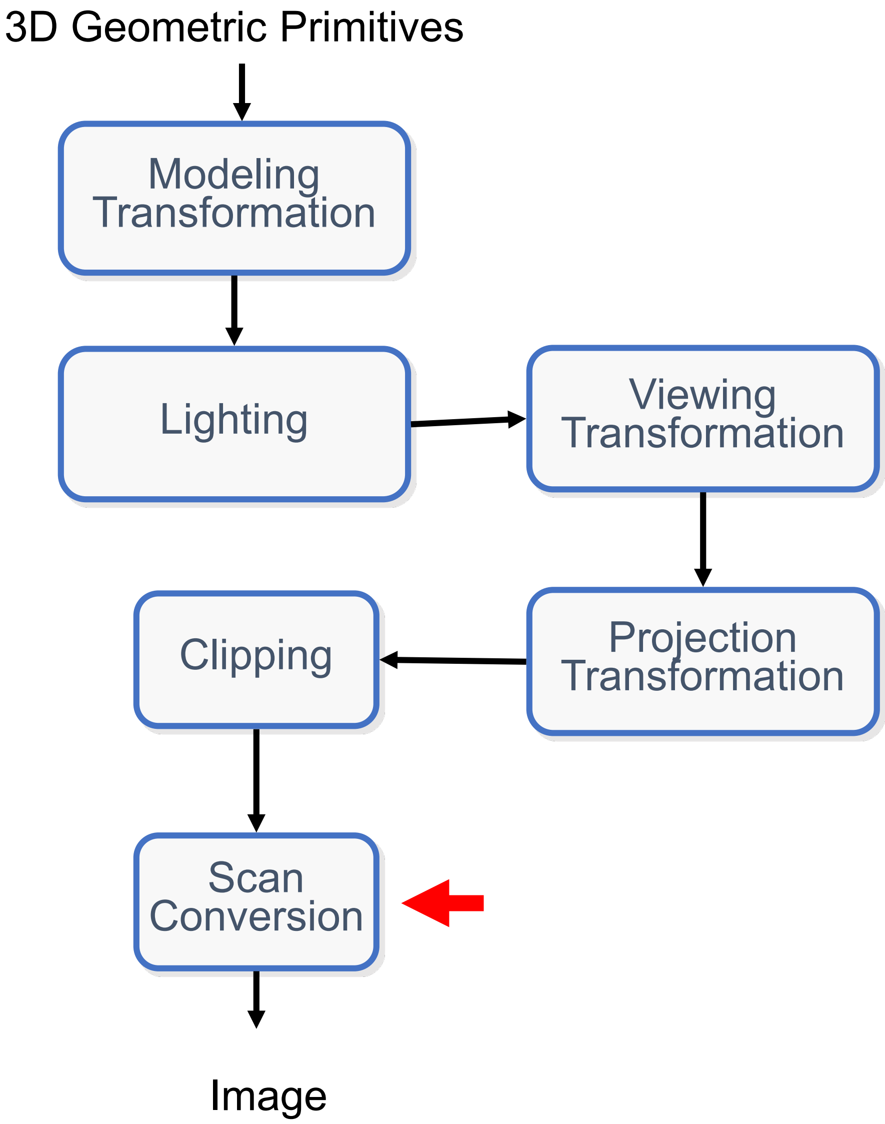
- Step 6 – Scan Conversion
  - What about hidden surfaces ?
  - We use the Z-buffer algorithm:
  - In addition to the frame buffer (keeping the pixel values), keep a Z-buffer containing the depth value of each pixel.
  - Surfaces are scan-converted in an arbitrary order. For each pixel (x,y), the Z-value is computed as well. The (x,y) pixel is overwritten only if its Z-values is closer to the viewing plane than the one already written at this location.



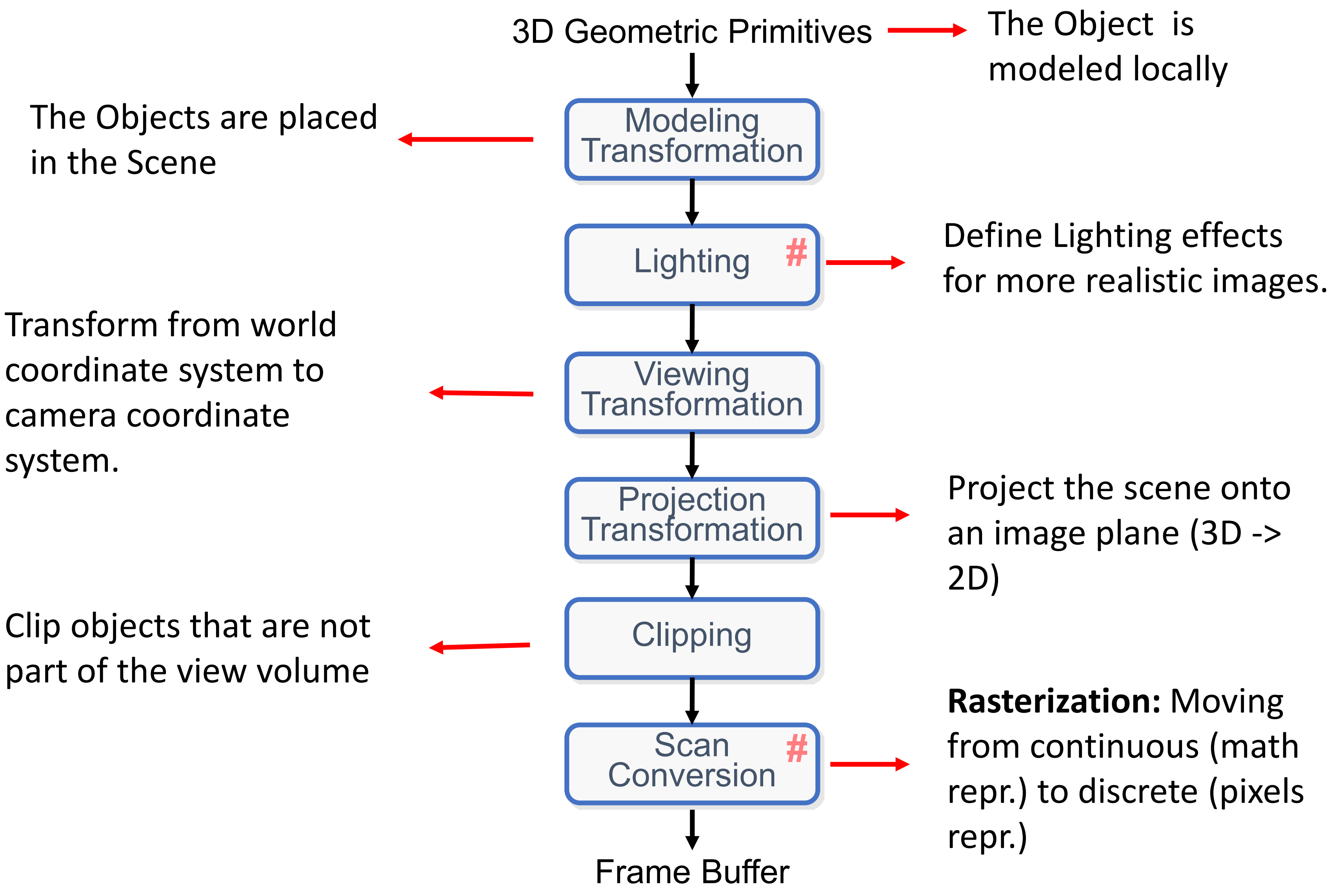
The Graphical Pipeline

- Step 6 – Scan Conversion
  - The Z-buffer algorithm – example:





Graphical Pipeline - Summary



Shadow Mapping in the Graphical Pipeline

- Assume a scene with one light source
- How shadows are casted in ray-tracing ?
  - Shoot rays from the intersection point to the light source:
    - If the ray intersect another object – the light is occluded by another object
      - There should be no light
    - Otherwise:
      - There is light – calcuate the intersection point color using Phong reflectance model
- But how can we support shadows in the graphical pipeline ?
  - Solution: shadow mapping

Shadow Mapping in the Graphical Pipeline

- If you looked out from a source of light:
  - All of the visible objects - appear in light.
  - Anything behind those objects, would be in shadow.

Shadow Mapping in the Graphical Pipeline

- Step (1) – Check which objects are closest to the light source
  - For each vertex (x,y,z) in the scene – calculate the coordinates in the "light" coordinate system
    - Simply multiply by the View transformation matrix in which the camera is located in the lightsource location
  - Project the vertex into 2D:
    - Perspective projection - if spotlight is used (point light)
      - What is the center of projection ?
    - Orthographic projection – if directional light is used
      - Whatis the direction of projection ?
  - Render the vertex*:
    - The z-buffer is extracted and saved in memory as shadow mapping
    - Avoid updating the color buffers and disable all lighting and texture calculations to save computation

Shadow Mapping in the Graphical Pipeline

- Step (2) – Check whether a point (x,y,z) is shadowed ?
  - Render the vertex (x,y,z) from the usual camera view
  - In the shading stage:
    - Transform the vertex (x,y,z) to the light coordinate system.
      - (x,y,z) -> (x',y',z')
    - Calculate the pixel coordinate of the vertex as if rendered from the light-source view
      - (x',y',z') -> (x'', y'', z'')
    - Check the shadow mapping (z-buffer from previous step) at (x'', y''):
      - The z-value stored in (x'',y'') is less than z':
        - The light source is occluded because the (x,y,z) is not closest to camera and shouldn't be included in the color calculation of (x,y,z)
      - Otherwise:
        - The light source is not occluded and can be considered in the color calculation of (x,y,z)

Shadow Mapping in the Graphical Pipeline

- In which stage shadow mapping should be considered ?
  - Lighting stage for Gouraud shading
  - Lighting and Rasterization stage in phong shading

Question

- We are given a 3D scene that is composed of several objects built from triangles.
- After we are done rendering it from the point of view of the camera and get a final image, we find out that we forgot to render one object in the scene.
  - The naïve way to fix this would be to render all the scene from scratch, but we want to fix the image in a more efficient way.

Question

- In each of the following algorithms explain if this is possible? If so, explain what is needed and if not explain why?

a) We are using simple Ray Casting algorithm (without shadows) to create the image.

- Ray Casting means no reflections or transparencies.
- Since we're not using shadows, the new object can't affect other objects.
- The pixel color is determined by the first intersection point of the ray that goes through that pixel.
  - Therefore, only pixels that object will appear on may be affected.
- Possible solution:
  - find the axis-aligned bounding box of the object and project it onto the picture.
  - These are the pixels that may change color.

Question

- In each of the following algorithms explain if this is possible? If so, explain what is needed and if not explain why?

b) We are using Ray Tracing algorithm to create the image.

- In Ray Tracing the new object can:
  - Project shadows on the other objects
  - Can be seen through other (reflective) objects in the scene
  - Can be reflected off any surface in the scene.
- This means there may be changes in all pixels.
  - We must re-render it the scene completely.

Question

- In each of the following algorithms explain if this is possible? If so, explain what is needed and if not explain why?

c) We are using the simple graphical pipeline algorithm (without shadows and textures) that uses a z-buffer to create the image.

- In the graphical pipeline we can simply continue the rendering process by sending the object triangles to the pipeline:
  - Use the z-buffer (with values from the original rendering) to resolve any object collision