**CG Note for Exam**

**Lecture1 Introduction**

1. What is Computer Graphics? The pictorial synthesis of real or imaginary objects from their computer-based 2D/3D models.
2. Applications of Computer Graphics:
   1. Video game
   2. Computer-aided design (CAD)
   3. Computer art
   4. Graphical User interface (GUI)
3. How Computer Graphics Works?
   1. Modeling
      1. 2D/3D Shape representation:
         1. Polygonal meshes (多边形网格)
         2. Subdivision surfaces (分割曲面)
         3. Splines (样条曲线)
         4. Geometry images
      2. Digital geometry processing: use relevant concepts for creation of 3D models.
         1. 3D scanning
         2. Surface reconstruction
         3. Denoising
         4. Simplification: Reduce the number of polygons in a surface while preserving the shape and details.
         5. Shape registration:
            1. Find the best alignment between two objects.
            2. Difficult because:

Different shape acquisition techniques

Some acquisition errors

* + - 1. Morphing: An interpolation technique; create a series of intermediate objects that change from one object to another.
  1. Animation
     1. Physics-based simulation (cloth simulation)
     2. Natural phenomena simulation (fire, fog, water, etc.)
     3. Motion capture
     4. Animation transfer (动画迁移): a correspondence map between the source and the target.
  2. Rendering: generate an image from a scene.
     1. Photo-realistic rendering: Simulate the real world, generate images that look very realistic.
     2. Non-photorealistic rendering
     3. Neural Rendering: combines machine learning techniques with physical knowledge. (e.g., NeRF)

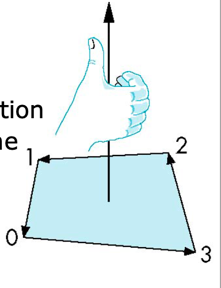
**Lecture2 Object Modeling**

1. 2D Drawing
   1. Convert the object into pixel pattern update the corresponding pixels in the frame buffer
   2. Drawing a Thin Line
      1. Set 𝑥 to 𝑥1 and 𝑦 to 𝑦1, and shade this pixel;
      2. Increase 𝑥 by 1, and correspondingly 𝑦= 𝑦 + 𝑚
      3. Compute D1 – distance of (*x*, *y*) from the center of the upper pixel;
      4. Compute D2 – distance of (*x*, *y*) from the center of the lower pixel;
      5. If D1 < D2, shade the upper pixel; otherwise, shade the lower pixel.
      6. if the endpoint 𝑥2, 𝑦2 is not achieved, turn to 2); otherwise, stop.
   3. Drawing a Circle:
      1. At each step, the path is extended by choosing the adjacent pixel which minimizes ;
      2. Can make use of the symmetrical property to save time.
   4. Flood-fill algorithm
      1. The border pixels of the region are set to the border color.
      2. A pixel which is completely inside the region is identified.
      3. If the current pixel has a value the same as the border color or the fill color, we stop here.
      4. Otherwise, we set the current pixel to the fill color.
      5. We recursively set the left, right, top and bottom pixels as the current pixel and repeat the checking process.
2. 3D Point Clouds
   1. Unstructured set of 3D point samples:
      1. Each point consists of geometry information (x, y, z) and other attributes.
      2. Acquired from range finder, computer vision.
      3. Used in autonomous driving
      4. Adv: Real-time acquisition; Connectivity information-free (无连接限制).

Dev: Difficult to perform geometry computation

* 1. Structured 3D Point Cloud-3D Point Clouds as 2D Images: The RGB values of each pixel correspond to the 3D coordinate of a point.

1. Polygon Meshes
   1. A polygon mesh *M* is a 3-tuple (**V, E, F**) that defines the shape of an object
      1. **V** is the set of vertices
         1. Each vertex must belong to at least one edge
      2. **E** is the set of edges
         1. Each edge must belong to at least one face
         2. An edge is a **boundary** edge if it belongs to only one face
      3. **F** is the set of faces
         1. Face can be assigned an orientation by defining the ordering of its vertices.
         2. counterclockwise order is the “front” side.



* 1. A mesh is orientable if all faces can be oriented consistently;

Not orientable: Möbius strip.

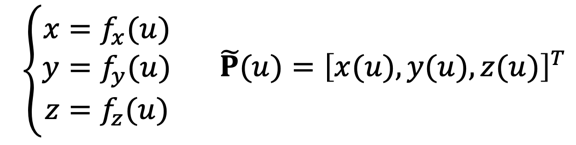
* 1. Euler Formula
     1. A convex polyhedron (a **closed** manifold mesh **without** holes/handles, 凸多面体)
     2. a **closed** **orientable** manifold mesh of genus **g** (number of holes in surface)
     3. a genus-*g* orientable manifold mesh with *b* boundaries
  2. Adv: Arbitrary geometry; Adaptive refinement; Efficient rendering.

Dev: Smooth deformation is difficult; Edges between polygons are noticeable.

1. Subdivision Surfaces (细分曲面)-Approximate the smooth surface by continuously refining the mesh. (通过不断细化网格来逼近光滑曲面)
2. Implicit Surfaces (隐式曲面)
   1. A procedure that can test to see if a point is on the surface. 𝑓(𝑥, 𝑦, 𝑧) = 0
   2. Adv: Efficient check whether point is inside (>0, outside; <0, inside)

Dev: Generating a point on the surface is difficult; Not easy to connect multiple implicit surfaces (拼接多个隐式曲面).

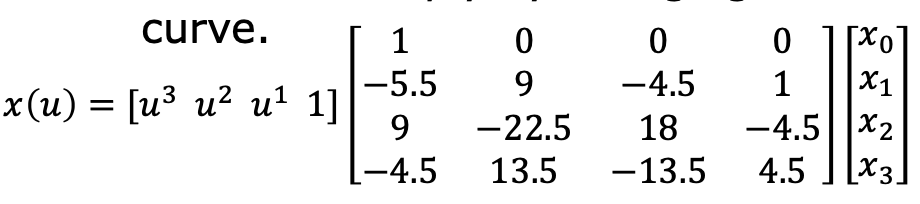
1. Parametric Surfaces
   1. Parametric curves

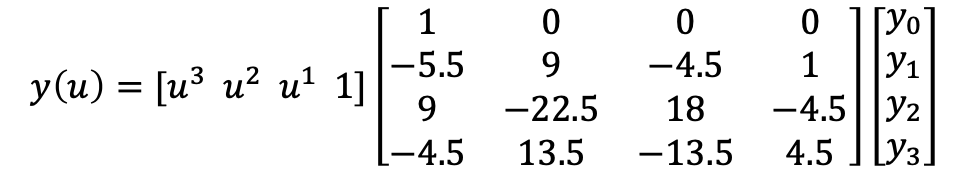


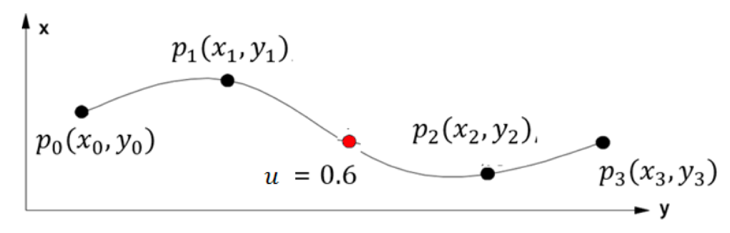
Cubic Parametric Polynomials (**三次参数曲线**)



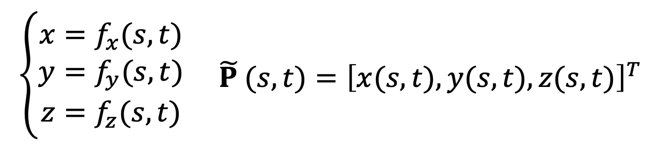
* + 1. P-geometry matrix, a set of control points for manipulating the shape of the curve.
    2. M-basis matrix, defines the basic properties of the curve.







* 1. \*Parametric Surfaces





* + 1. P-geometry matrix, a set of control points for manipulating the shape of the curve.
    2. M-basis matrix, defines the basic properties of the surface.
  1. Adv: Easy smooth deformation (平滑变形).

Dev: Expensive to render.

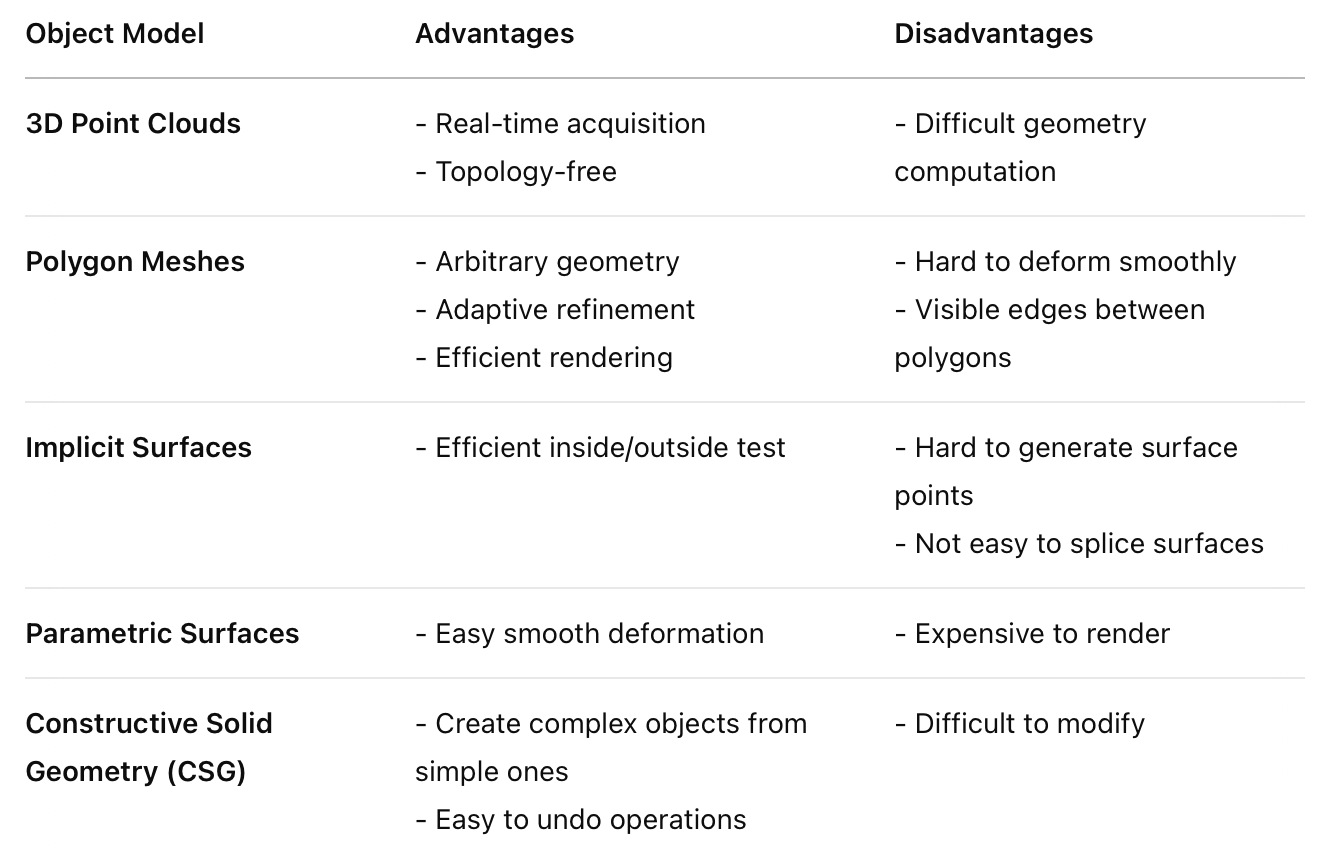
1. Voxel Representation (体素表示)
2. Constructive Solid Geometry (CSG) (构造实体几何)
   1. Use Boolean operations to combine simpler objects.
   2. Adv: Complex objects can be created with simple ones; easy to undo operations.

Dev: Modification would be difficult.

1. Fractals (分形):
   1. objects which are self-similar.
   2. Generate by recursively applying the same transformation function on a given object.

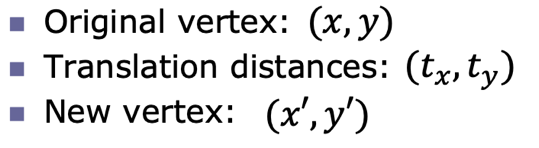
𝑥 = 𝑓(… 𝑓(𝑥))

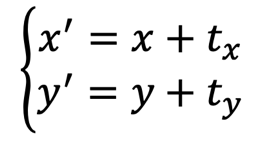
* 1. May introduce some random factors to make generated objects more realistic.



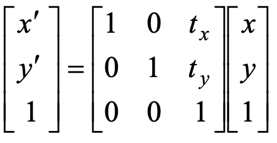
**Lecture 3 Geometric Transformation**

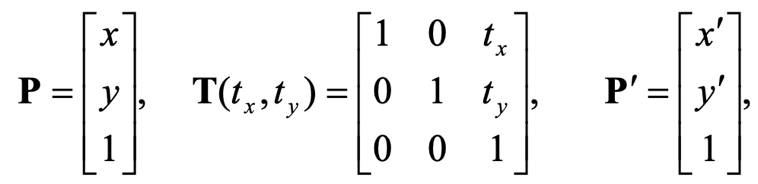
1. 2D Transformations
   1. Translation
      1. Normal Formula

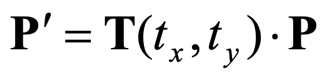




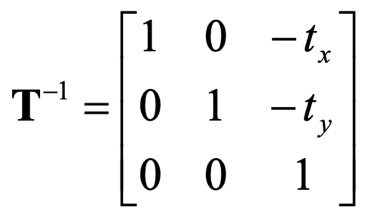
* + 1. 2D Translation Matrix



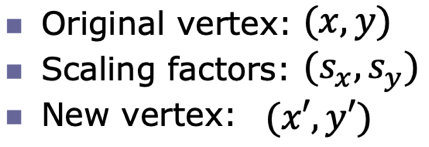


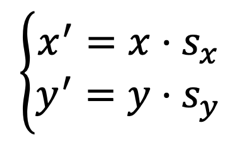


Inverse of the translation matrix

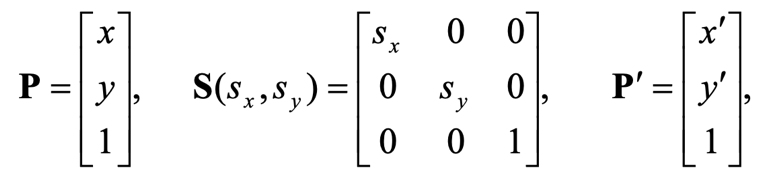


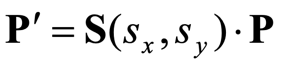
* 1. Scaling
     1. the reference point is the **origin**



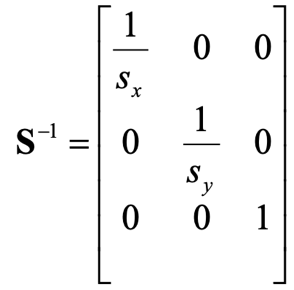


2D Scaling Matrix:

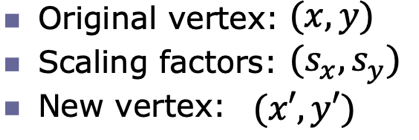


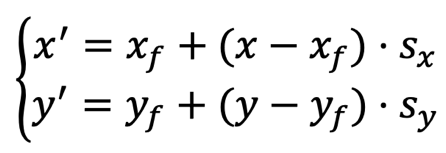


Inverse of the scaling matrix:

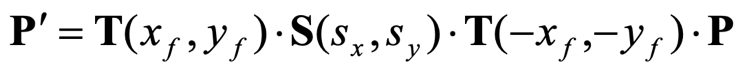


* + 1. **arbitrary** reference point (𝑥𝑓, 𝑦𝑓)

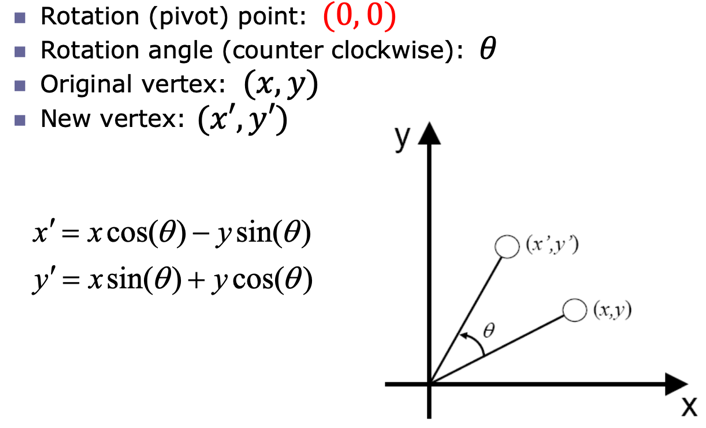




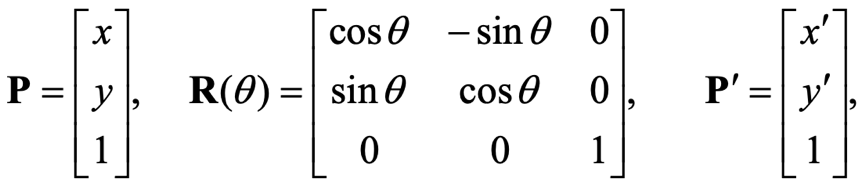
2D Scaling Matrix:



* 1. Rotation
     1. Rotation by an angle 𝜃 in CW is equivalent to the rotation by an angle 2𝜋 − 𝜃 in CCW.
     2. The pivot point is the **origin**

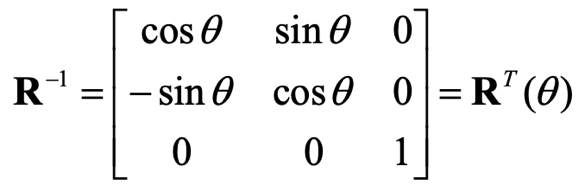


2D Rotation Matrix:

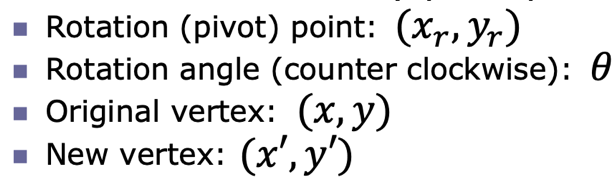


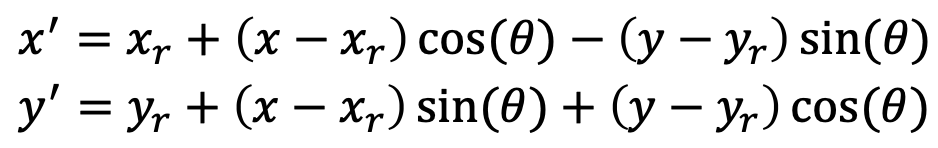


Inverse of the rotation matrix:



* + 1. arbitrary pivot position

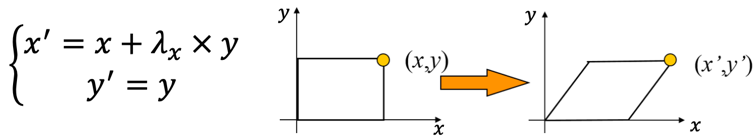


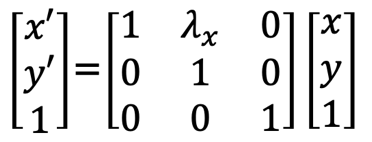


2D Rotation Matrix:

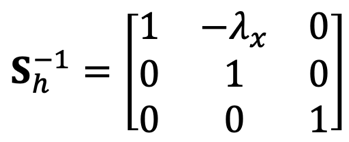


* 1. Shearing
     1. A horizontal shearing (or 𝑥-shearing)

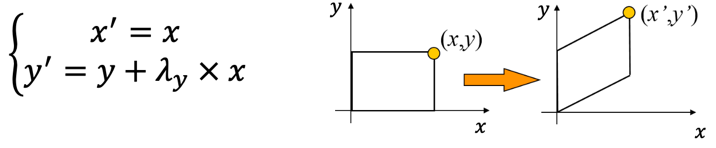


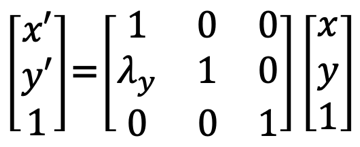


Inverse of the shearing matrix

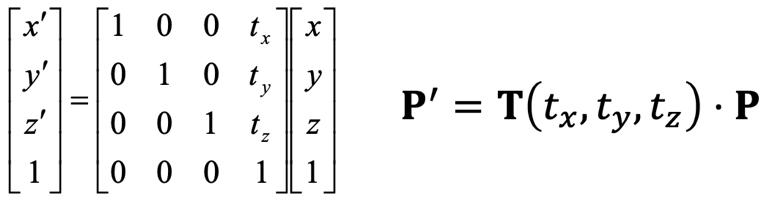


* + 1. A vertical shearing (or 𝑦𝑦-shearing)

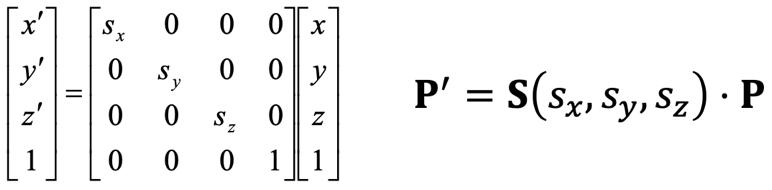




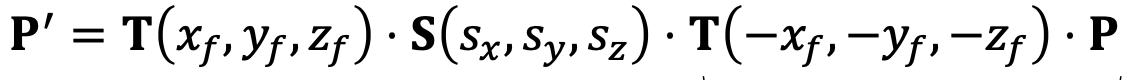
1. 3D Transformations
   1. Translation



* 1. Scaling
     1. Scaling relative to the **origin**



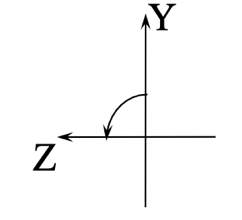
* + 1. Arbitrary fixed point (𝑥𝑓, 𝑦𝑓, 𝑧𝑓)

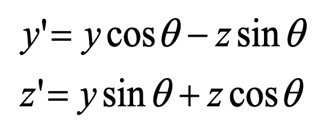


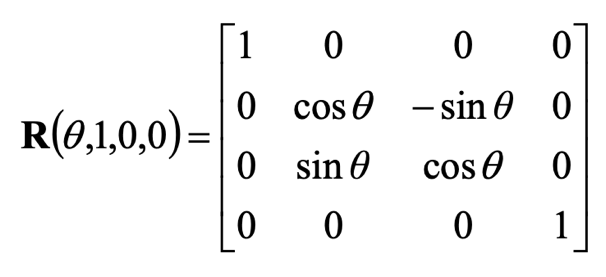
* 1. Rotation
     1. Rotation about a coordinate axis:

当你沿着坐标轴的正方向看向原点时，旋转是逆时针的。

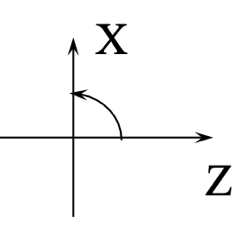
* + 1. Rotation around 𝑥*-* axis——y上z下

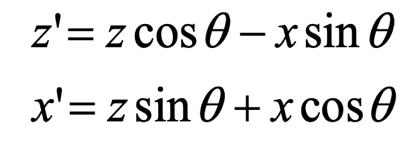


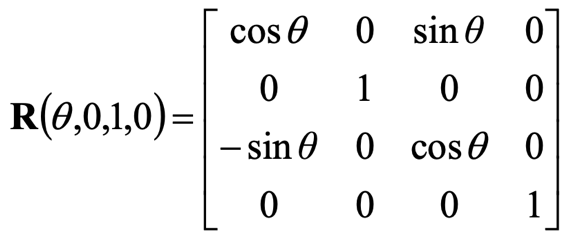




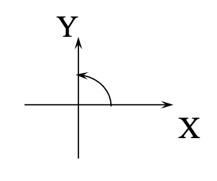
* + 1. Rotation around 𝑦-axis——z上x下

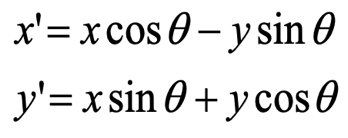


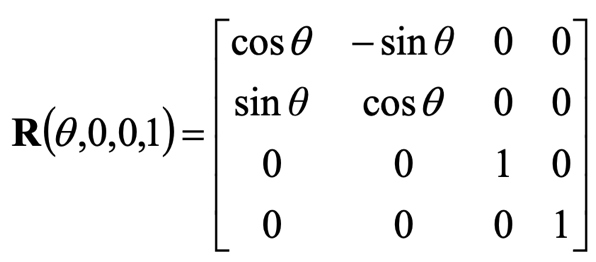




* + 1. Rotation around 𝑧*-*axis——x上y下

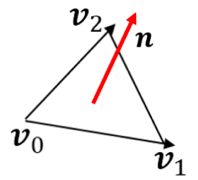






**Lecture 4 Hidden Surface Removal Illumination & Shading**

1. Hidden Surface Removal
   1. Def: Determine what is visible within a scene from a chosen viewing position.
   2. Object space algorithms: Determine which 3D objects are in front of others.
      1. Back-face culling algorithm
         1. Draw polygons facing the camera and hide the ones which are facing away from viewpoint.
         2. polygonal face 𝑓, viewpoint 𝒗, point 𝒑, face normal **n**.
            1. Invisible if 𝒏· (𝒑 − 𝒗) >= 0
            2. Visible if 𝒏· (𝒑 − 𝒗) < 0
            3. Calculate the normal of a plane: 𝒏 = (𝒗1− 𝒗0) × (𝒗2− 𝒗0)

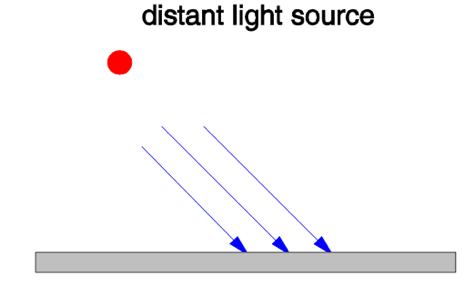


* + - 1. Adv: Implement easily; efficient.

Dev: fail for concave polyhedral (凹多面体).

* 1. Image space algorithms: Determine what color is used to paint at each pixel.
     1. Depth-sorting algorithm
        1. Process:
           1. Sort all surfaces according to their distances from the view point.
           2. Render the surfaces to the frame buffer one at a time starting from the ***farthest*** surface.
           3. A surface drawn later will replace overlapping surfaces drawn earlier.
           4. After all the surfaces have been processed, the frame buffer stores the final image.
        2. A few objects, efficient; large amount of objects, complex and time consuming.
        3. Not processed correctly: intersecting polygons or cyclic overlap.
     2. Depth-buffering algorithm
        1. Two buffers: a frame buffer and a depth buffer.
           1. Depth buffer: the same resolution as the frame buffer; store the depth values of surfaces.
        2. Process:
           1. Initialize the depth buffer and frame buffer to very large positive values and the background color.
           2. Surfaces are rendered one at a time at random.
           3. The depth value of each pixel of a surface is calculated.
           4. Depth value < corresponding depth value Use the depth value and the color value of this surface to update the depth value in the depth buffer and the color value in the frame buffer

1. Illumination
   1. Light sources
      1. Ambient light (环境光): a uniform intensity from all directions.
      2. A point light source: It emits directional light rays.



* + 1. Spotlight (**聚光灯**)
    2. An Area light source (**面光源**)
  1. Illumination Model: determine the color of a surface point by simulating some light attributes.

Phong illumination model consists of three parts:

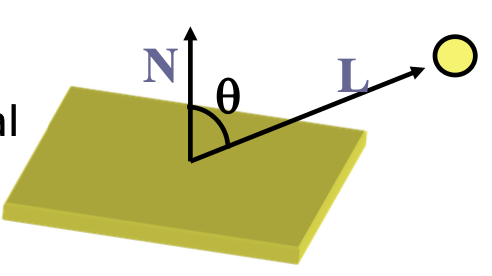
* + 1. Ambient reflection (环境反射)

𝐼𝑎 = 𝑘𝑎 ∗ 𝐼𝑐

* + - 1. 𝐼𝑎: Ambient light intensity, equal in all directions.
      2. 𝑘𝑎: an ambient reflection coefficient, 0 < 𝑘𝑎 < 1.
      3. Determined by: Surface properties; independent of: surface’s position and orientation.
    1. Diffuse reflection (漫反射)

𝐼𝑑 = 𝑘𝑑 ∗ 𝐼𝐿 ∗ cos 𝜃= 𝑘𝑑 ∗ 𝐼𝐿 ∗ (𝐍 ⋅ 𝐋)

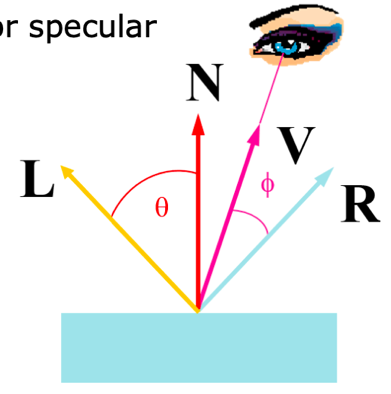
* + - 1. 𝑘𝑑: surface diffuse coefficient.
      2. 𝜃: the angle between the surface normal and the incoming light ray.



* + 1. Specular refection (镜面反射)

𝐼𝑠 = 𝑘𝑠 ∗ 𝐼𝐿 ∗ cos𝑛 𝜙= 𝑘𝑠 ∗ 𝐼𝐿 ∗ (𝐕 ⋅ 𝐑) 𝑛

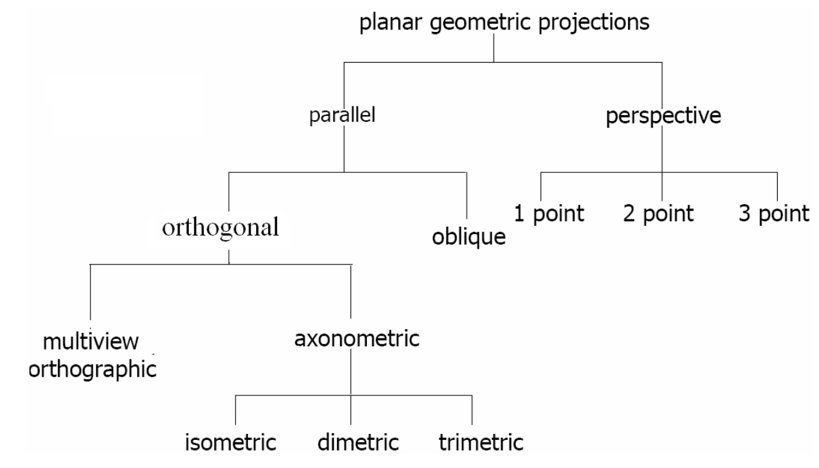
* + - 1. 𝑘𝑠 : surface specular reflection coefficient.
      2. 𝑛: 高光指数，决定高光的集中程度（镜面反射越强，n越大，n=∞代表理想镜面）



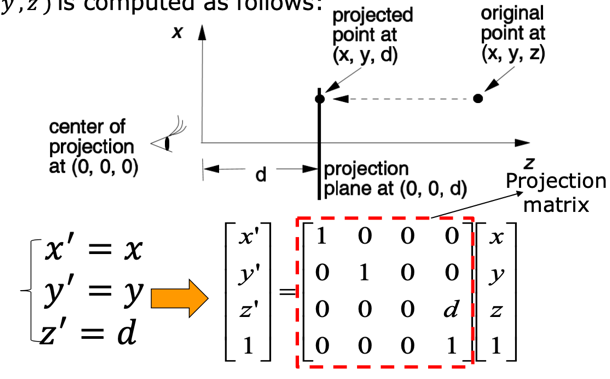
1. Shading: Use the color calculation to determine the pixel colors at all projected positions.
   1. Flat shading:
      1. The whole polygon is shaded with the same intensity value.
      2. Fast and simple but causing intensity discontinuity.
   2. Smooth shading
      1. Gouraud shading
         1. Interpolate color across triangles.
         2. Fast but not properly handle specular highlights.
      2. Phong shading
         1. Interpolate normals across triangles
         2. More accurate but slower.

**Lecture 5 Projection and Clipping**

1. Projection



* 1. Parallel projection
     1. Two types of parallel projections
        1. Orthographic projection (正投影): projectors are perpendicular to (垂直于) projection plane.



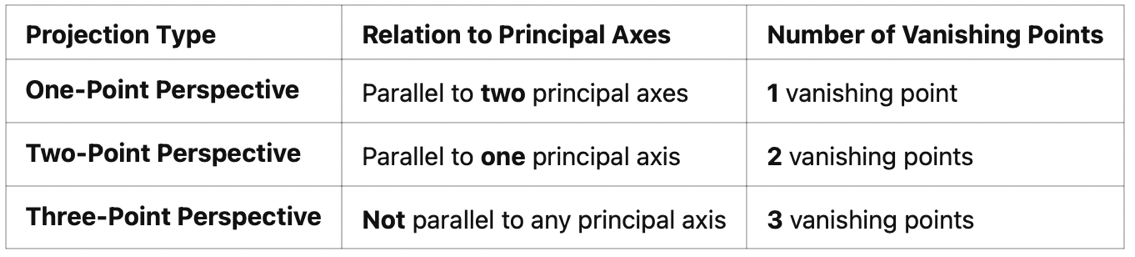
* + - * 1. Multi-view Orthographic Projection: The projection plane is parallel to the coordinate plane; Three-dimensional object projected onto the six sides of the "box".
        2. Axonometric Projection (轴测投影):

**Trimetric**: angles between the three principal axes are different.

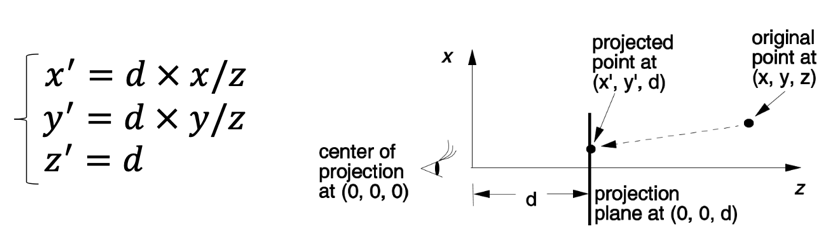
**Dimetric**: angles between two of the principal axes are equal.

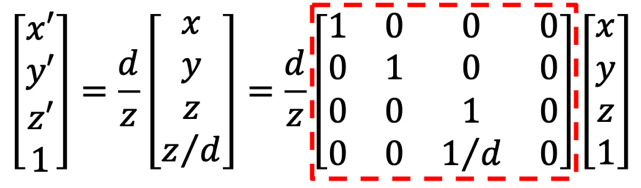
**Isometric**: angles between the three principal axes are equal.

* + - 1. Oblique projection (斜投影): Not perpendicular to the projection plane.
  1. Perspective projection
     1. Vanishing Points: Any set of parallel lines that are not parallel to the projection plane will converge to a vanishing point.
     2. Three Types:

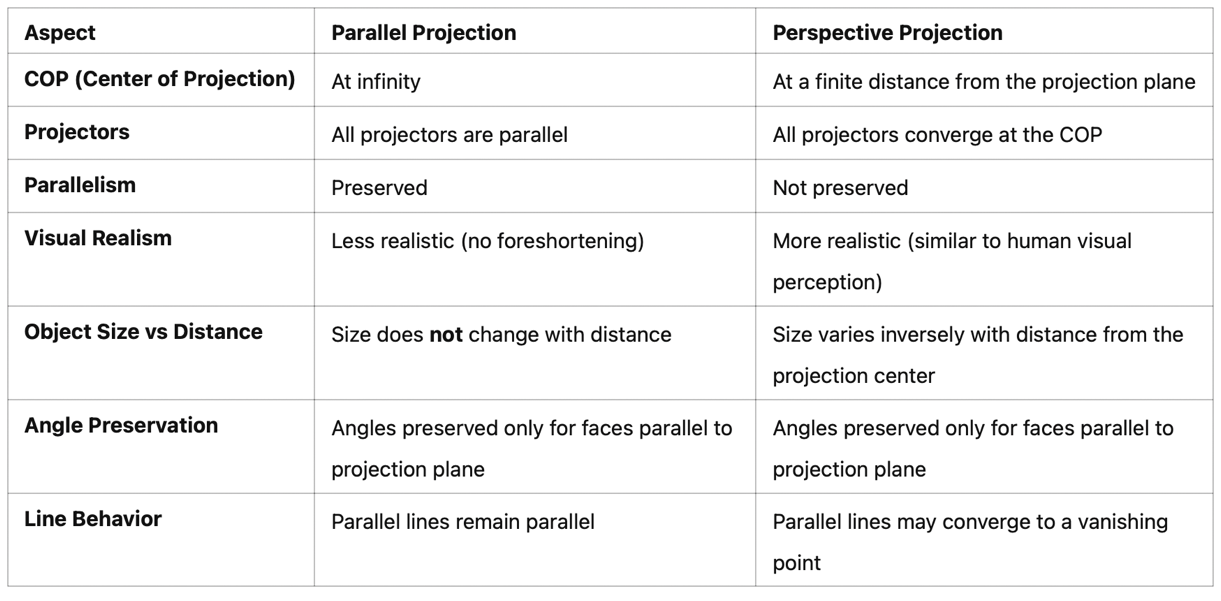


* + 1. Perspective projection matrix

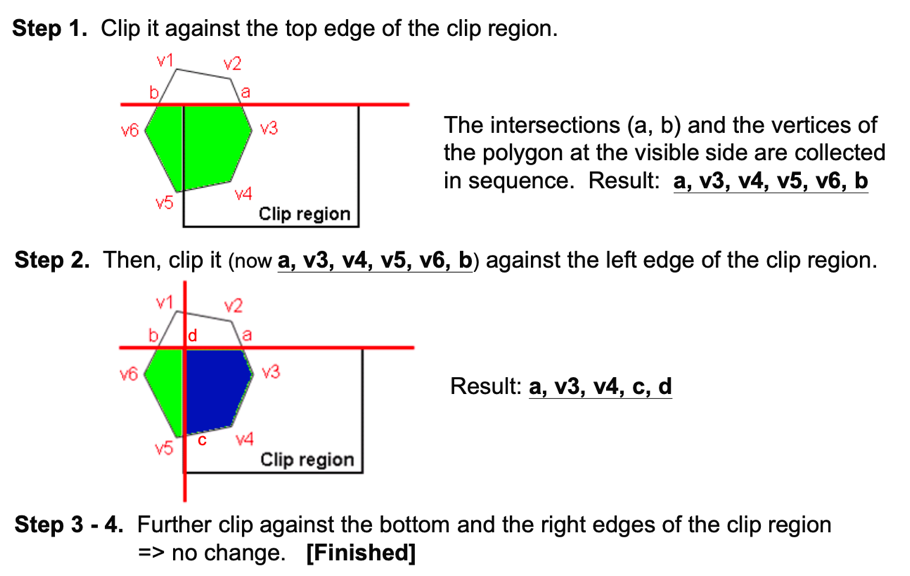


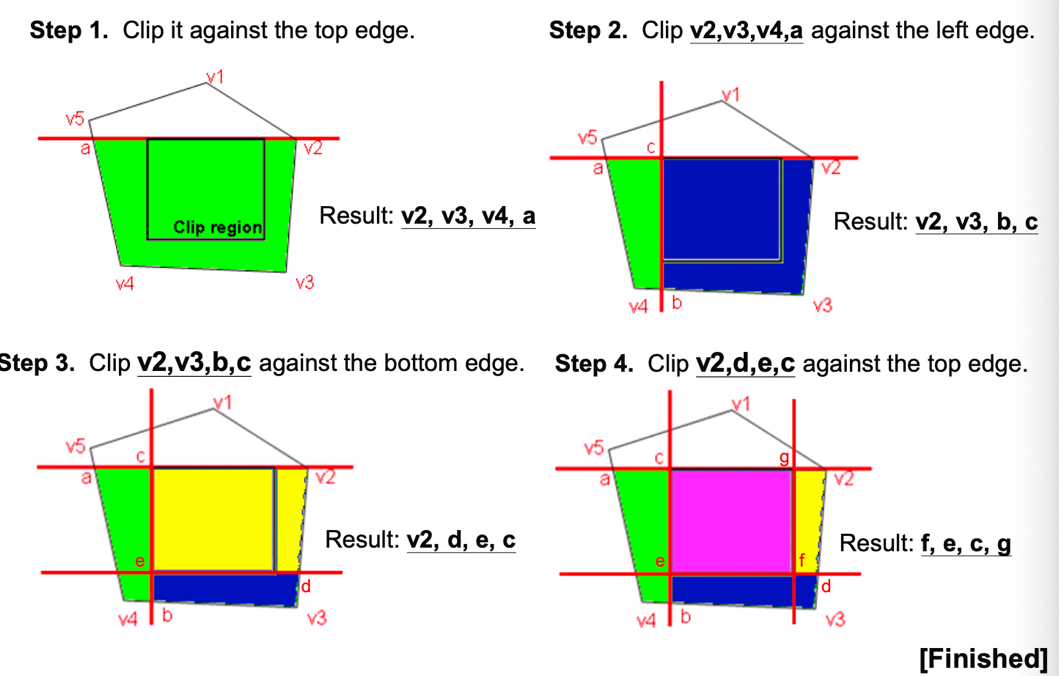


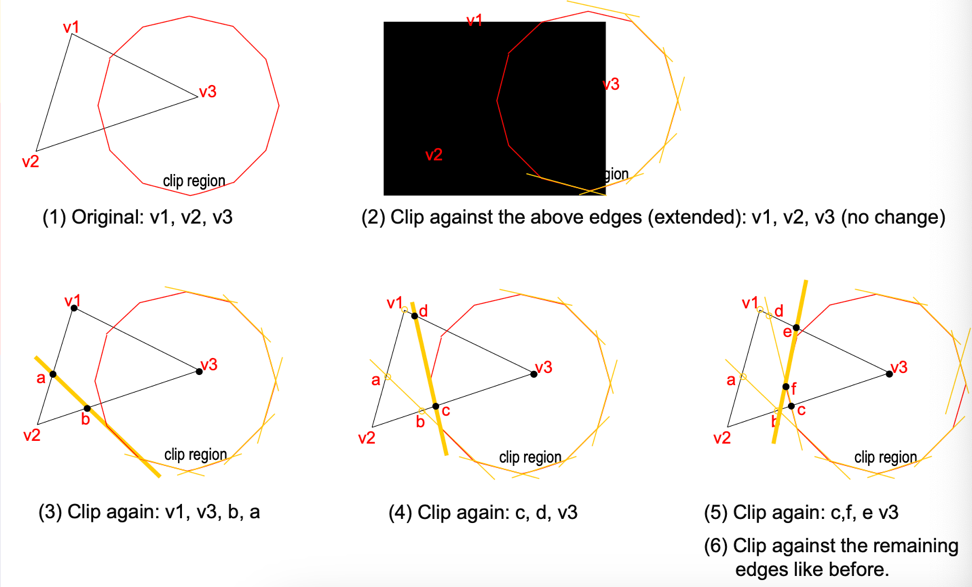
* 1. Perspective vs. Parallel Projection



1. Clipping: a process to determine the portion of an object lying inside (or outside) a clip region.







**Lecture 6 Aliasing and Anti-aliasing**

1. Aliasing (混叠)
   1. The Sampling Theorem: to adequately capture a function with maximum frequency 𝐹, we need to sample it at frequency 𝑁 = 2𝐹, where 𝑁 is called the *Nyquist limit*.
   2. **The Aliasing Problems in CG:** a straight line looks like staircases and a character look like discretized in 2D drawing.
      1. Spatial aliasing (空间混叠):
         1. In static images.
         2. Reason: the point sampling nature (coarse approximation, 粗近似).
      2. Temporal aliasing (时间混叠):
         1. In video sequences.
         2. Two main reasons:
            1. Spatial aliasing.
            2. Undersampling (欠采样) in time domain.
2. Anti-aliasing(抗锯齿) in CG
   1. Supersampling
      1. Increases the number of samples.
      2. Process:
         1. Create a virtual image at higher resolution than the final image.
         2. Apply a low-pass filter (低通滤波器).
         3. Resample the filtered image.
      3. Adv: Easy to implement.

Dev: Considerable memory and time; Cannot eliminate aliasing.

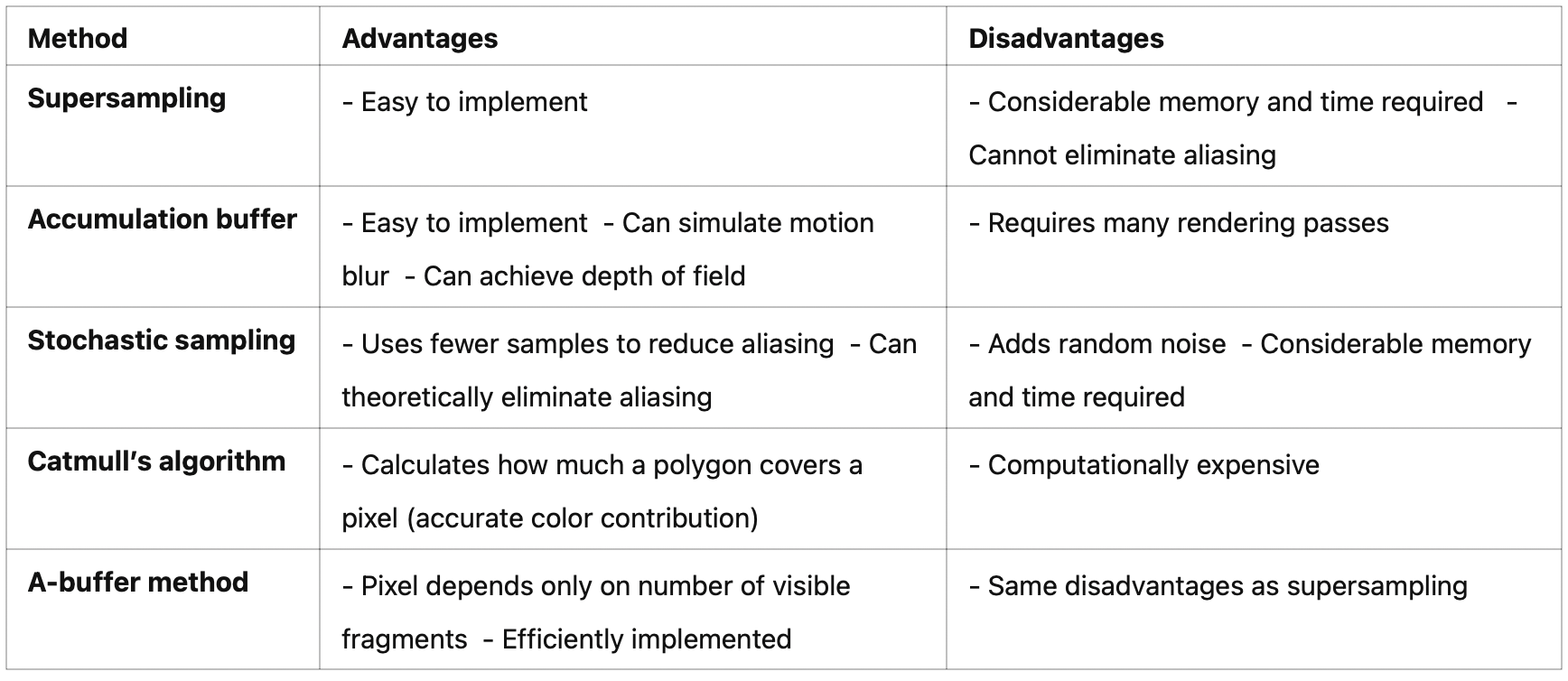
* 1. Accumulation buffer (累积缓冲)
     1. Renders one subpixel at each pass.
     2. Adv: Easy to implement; Can simulate motion blur; Can achieve depth of field.

Dev: Use many rendering passes.

* 1. Stochastic sampling (随机采样)
     1. Use random sample points in each pixel.
     2. Two kinds:
        1. Poisson-disc distribution（泊松盘分布）: Keep a random distribution of minimum distance between points.
        2. Jittered distribution（抖动分布）: Based on a regular grid, random perturbation (随机扰动) is carried out for each sampling point.
     3. Adv: Use less samples to reduce aliasing; theoretically eliminate aliasing.

Dev: Add random noise; Considerable memory and time.

* 1. Catmull’s algorithm
     1. Adv: Calculate how much a polygon covers a pixel.
     2. Dev: Large computation.
  2. The A-buffer method
     1. Adv: Pixel depends only on number of visible fragments; Implemented efficiently.
     2. Dev: the same as supersampling algorithm.

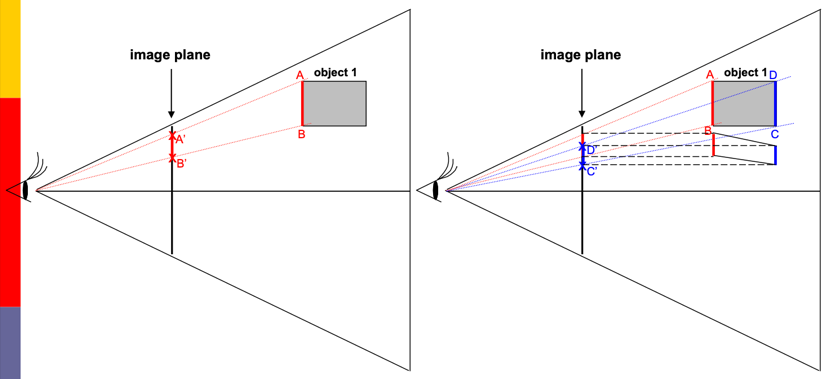


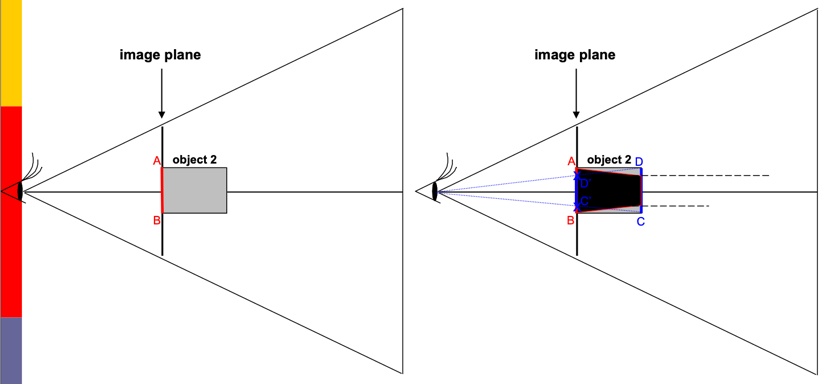
**Lecture 7 The Rendering Pipeline**

1. Def: a continuous object is processed as a collection of discrete pixels
2. Process:
   1. Input object models.
      1. Created using a program.
      2. Following properties:
         1. Be composed of primitives.
         2. Primitives need to be transformed.
         3. Primitives are sent to the pipeline one by one randomly.
         4. In local coordinate.
   2. World coordinate transformation
   3. Perspective transformation
      1. Formula:

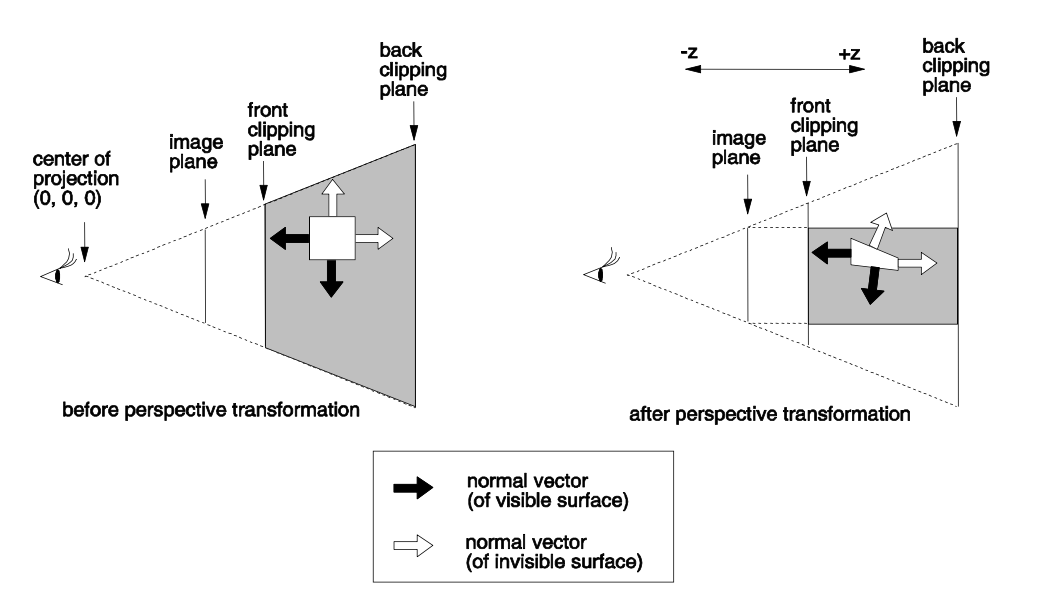


* + 1. Perspective transformation vs. Perspective projection: transformation retains the depth value of vertex, while projection does not.
    2. After perspective transformation, the projection becomes a parallel projection.
    3. Exercise:





* 1. Back-face removal
     1. After perspective transformation, If the 𝑧 component of the normal vector is positive, it is a back face. If it is negative, it is a front face.



* 1. Clipping:
     1. remove primitives outside the view volume.
     2. Before perspective transformation, this operation will have to clip primitives against arbitrary planes. (针对任意方向的裁剪平面来处理图元)

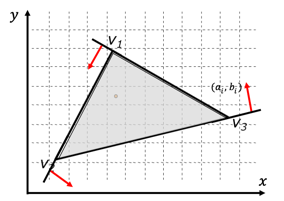
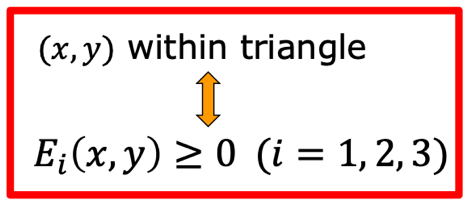
After perspective transformation, straightforward and can be performed in 2D.

* + 1. The origin is at the center of projection, the six clipping planes after perspective transformation becomes:

𝑥 = – 𝐴, 𝑥 = 𝐴, 𝑦= – 𝐵, 𝑦= 𝐵, 𝑧 = -C, 𝑧 = 𝐶

* 1. Rasterization:
     1. take a primitive and figure out which pixels it covers.
     2. E: three half-spaces defined by these lines.

𝐸𝑖 (𝑥, 𝑦) = 𝑎𝑖𝑥 + 𝑏𝑖𝑦 + 𝑐𝑖 (𝑖 = 1, 2, 3)



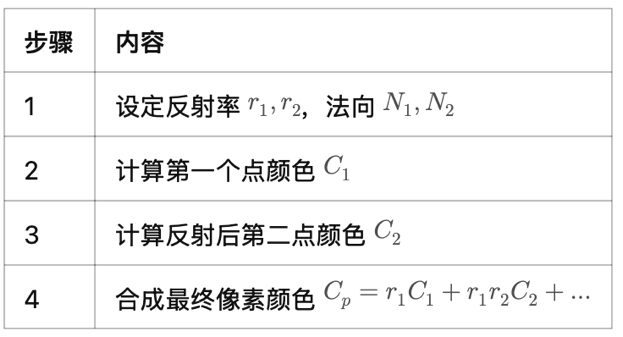
* 1. Hidden surface removal and shading: removes primitives obscured by other primitives.
  2. Output image: Use the color calculation to determine the pixel colors.

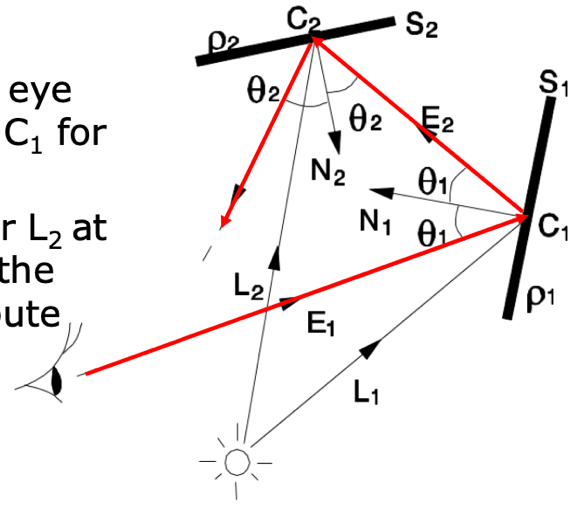
**Lecture 8 Computer Graphics** **Ray Tracing and Radiosity**

1. Two Rendering Approaches
   1. Start from geometry:
      1. fast
      2. hard to compute accurate shadows, reflections and refractions
   2. Start from pixels
2. **Ray Tracing**
   1. Generates an image by tracing the flow of rays in a scene.
   2. In a **backward** manner-from the eye point to the light sources.
   3. Process:
      1. The Primary Ray starts from the viewpoint, passes through each pixel, enters the scene and intersects with the first object.
      2. After obtaining the intersection point, the color of the point is calculated based on the normal vector and light source information of that point.
      3. Shadow detection: If there is an obstruction between a light source and an intersection point, the point is a shadow area and the contribution of the light source is not considered.

Recursive tracking: If the surface is reflective or transparent, continue to track the reflected rays and refracted rays.

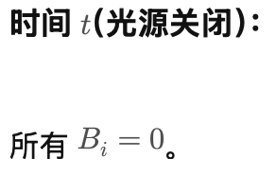
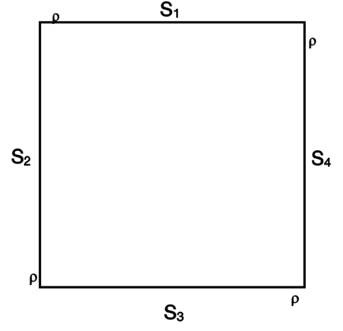
* + 1. After each reflection/refraction, continue to determine the new intersection point and recalculate the color again until:
       1. Reach a certain number of reflections.
       2. The color contribution is too small.
    2. The final color of each pixel is a weighted combination of the colors of multiple intersection points.
  1. **Calculate the color of a pixel**

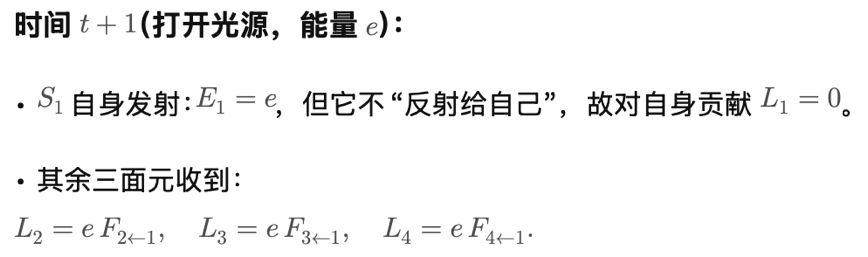
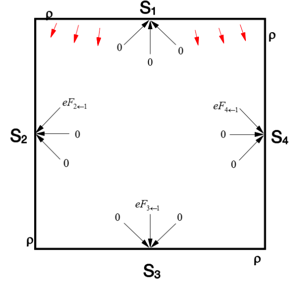


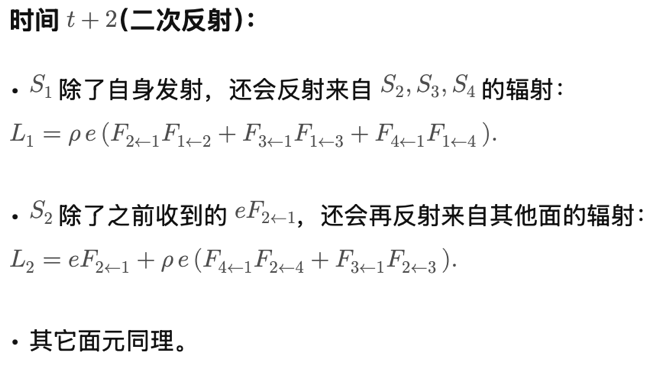
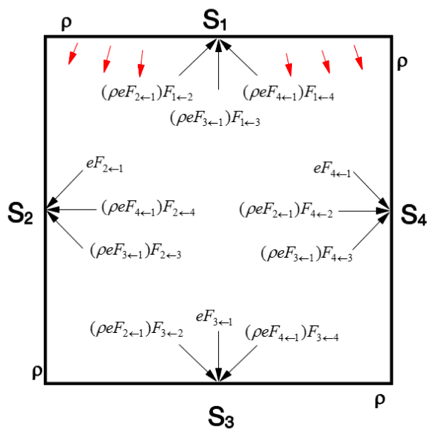


* 1. Adv:
     1. Consider direct **specular** reflections, direct diffuse reflections and indirect **specular** reflection;
     2. Consider light refraction and shadowing.
  2. Dev:
     1. Large number of ray-surface intersection tests and color calculations;
     2. Not consider indirect diffuse reflections.
  3. **Ray Tracing Acceleration**
     1. Bounding volumes (包围体)
        1. a bounding volume is constructed for each object in the scene.
        2. first check if the ray intersects
        3. Further speedup needs a hierarchical bounding volume of objects.
     2. Space subdivision (空间划分)
        1. Uniform space subdivision
           1. Divide the space into regular voxels (网格)
           2. Each voxel(立方体体素) stores a list of objects that intersect with it.
           3. When light passes through these Voxels, only whether the objects stored in them intersect is detected, rather than all the objects in the scene.
        2. Non-uniform space subdivision: Subdivides the scene hierarchically to obtain fewer voxels.

1. **Radiosity** (辐射度法)
   1. Aim: modeling **diffuse reflection** among the surfaces in the scene.
   2. Process:

* 1. Convergence
     1. As 𝜌 is normally smaller than 1, the subsequent terms will become smaller and smaller due to the increasing number of reflections.
     2. The emitted energy should converge to a certain value.
  2. Large surface is divided into small patches.
  3. The radiosity method cannot handle **specular reflection**

**Lecture 9 Real-Time Rendering**

1. **Progressive Rendering**
   1. produce multiple levels of detail (LoDs)
      1. Discrete LoD
         1. Pre-compute a few models at different resolutions.
         2. Adv: efficient.

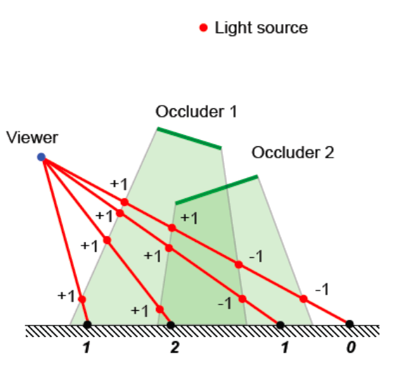
Dev: hard to determine number of LoDs; Visual artifacts (视觉伪影) when switching models.

* + 1. Progressive mesh
       1. two operations:

Edge collapse can reduce resolution; vertex split is the reversed operation.

* + - 1. Use progressive mesh structure to store the model in different resolution.
      2. Efficient but not effective for large models.
    1. Selective refinement (For large model)
       1. selectively refine the model resolution of local region interested at while keeping the rest at low resolution.
       2. Difficulty: Strong dependency between neighboring regions of triangles.

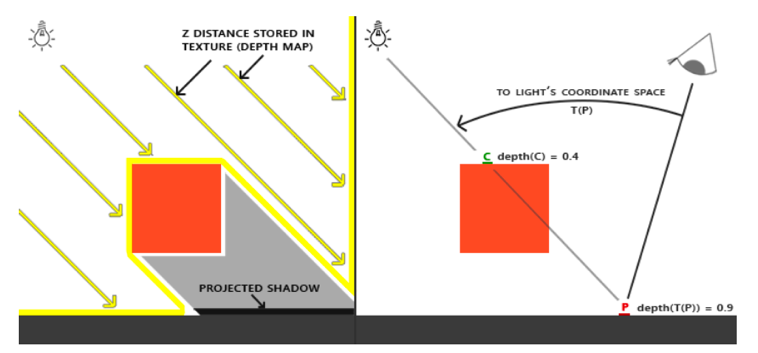
1. **Estimation**
   1. **Visual Quality Estimation**
      1. Measured based on:
         1. LoDs.
         2. Distance from the viewer.
         3. Size on the screen.
      2. Other factors affect the quality perception:
         1. Moving speed
         2. Angular distance from the viewer’s line of sight.
   2. **Rendering Cost Estimation-**a method to estimate the cost of rendering each object: At whatever level of detail; Use whatever rendering method.
2. **Shadows**
   1. provide information regarding the geometric relationship and the light sources.
   2. Hard shadows:
      1. Only umbra regions (本影, 完全没有光照射到的区域).
      2. Point light sources.
      3. Generation of hard shadows:
         1. Shadow volume methods: more complex because of handling additional geometry.
            1. From the view point, we join a line to each point of interest.
            2. We start a counter from zero.
            3. Whenever the line crosses a front-facing polygon of a shadow volume, we add 1 to the counter.
            4. Whenever the line crosses a back-facing polygon of a shadow volume, we subtract 1 from it.
            5. A point is in shadow if the final counter value is a positive number.



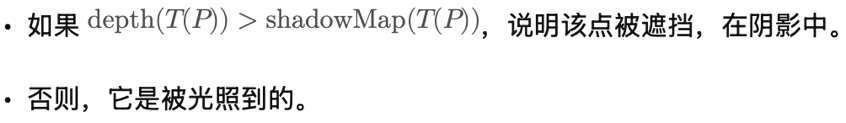
* + - 1. Shadow map methods
         1. Process:

Depth map: Each pixel stores the depth from the light source to that point.

对于观察者视角下的每一个像素P，把它的三维位置转换成光源视角下的位置T(P)，然后取得它的深度值depth(T(P)) 。



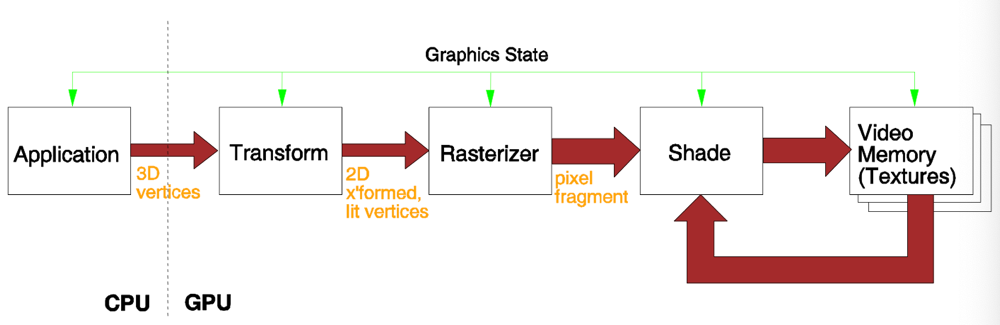
比较当前像素在光源坐标下的深度 depth(T(P))  和 shadow map 中的深度：



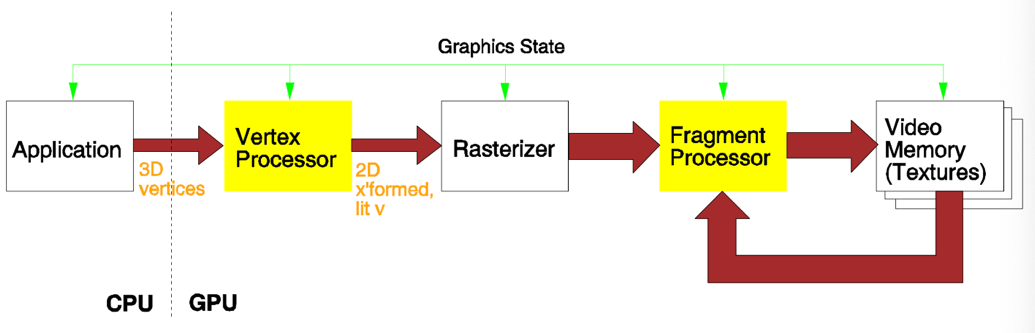
* + - * 1. The standard method has the problem of aliasing, which can be improved by perspective projection,
  1. **Soft Shadows**
     1. Contain umbra regions and penumbra regions (半影, 部分被遮挡、部分能见到光源的区域，亮度介于明亮区和本影之间).
     2. Area light sources.
     3. Generation of soft shadows
        1. Combining point light sources
        2. Generation of soft shadows

**Lecture 10 GPU and Computer Animation**

1. **CPU vs. GPU**
   1. CPU:
      1. Good at control-heavy tasks not data-heavy tasks
      2. Few arithmetic units
      3. Optimized for low latency
   2. GPU:
      1. Simple control
      2. Lots of calculations
      3. Latency-tolerant
2. **Modern GPUs**
   1. highly programmable
   2. high precision computations
3. **GPU Architecture**
   1. Traditional:



* 1. Advanced hardware graphics pipeline:
     1. Programmable vertex processors
        1. Transformation
        2. Back-face removal
     2. Programmable pixel processors
        1. Clipping
     3. Fragment processors: Depth comparison.



1. **Efficient Computation**
   1. Data-level parallelism
   2. Task-level parallelism
2. **Efficient Communication**
   1. The performance of GPU parallelization is limited by the complexities of CPU-GPU communication.
   2. Three ways to improve:
      1. Transmit the entire data stream
      2. Structuring applications as pipelines of kernels (将程序结构化为“核函数”流水线)

理解： 如果多个计算核之间可以直接传递数据，而不需要把中间结果写回内存，会极大提升效率。

* + 1. Favor deep pipelining, i.e., maximizing the number of pipeline stages

理解： 就像工厂流水线越长越细，每个阶段可以同时处理不同数据，这样数据等待内存时不会阻塞后续操作。

1. **Computer Animation**
   1. Artist-directed (e.g., keyframing, 关键帧设定): Specify importation events and computer fills in the rest via interpolation.
   2. Procedural (程序驱动) (e.g., simulation): Define a set of programs or operations to generate animations; Each operation can generate or modify data and can be performed conditionally or unconditionally; Run the simulation by defining rules, initial conditions, and parameter values.
   3. Data-driven (e.g., motion capture): sample and record motion of humans.
2. **\*Current MoCap Technologies**
   1. Electromagnetic mocap:
      1. Adv: no occlusion problems; no special lighting condition.
      2. Dev: cannot capture facial expression; hard to capture small bone movement.
   2. Electromechanical mocap
      1. Adv: free-of-occlusion; portable.
      2. Dev: cannot capture facial expression; hard to capture small bone movement.
   3. Optical mocap
      1. Adv: most accurate; high frame rate.
      2. Dev: occlusion problems; expensive.
   4. Markerless mocap: Video-based mocap; Depth sensor-based.
3. **\*Optical MoCap Pipeline**: PlanningCalibration Processing markers Data processing

**Question in Exam**

1. In the lecture, we have learned the line clipping algorithm, in which the clipping

region is a rectangular in 2D space. Please show the **main** idea for extending this

algorithm in 3D space, i.e., clipping a line against a 3D cubic.

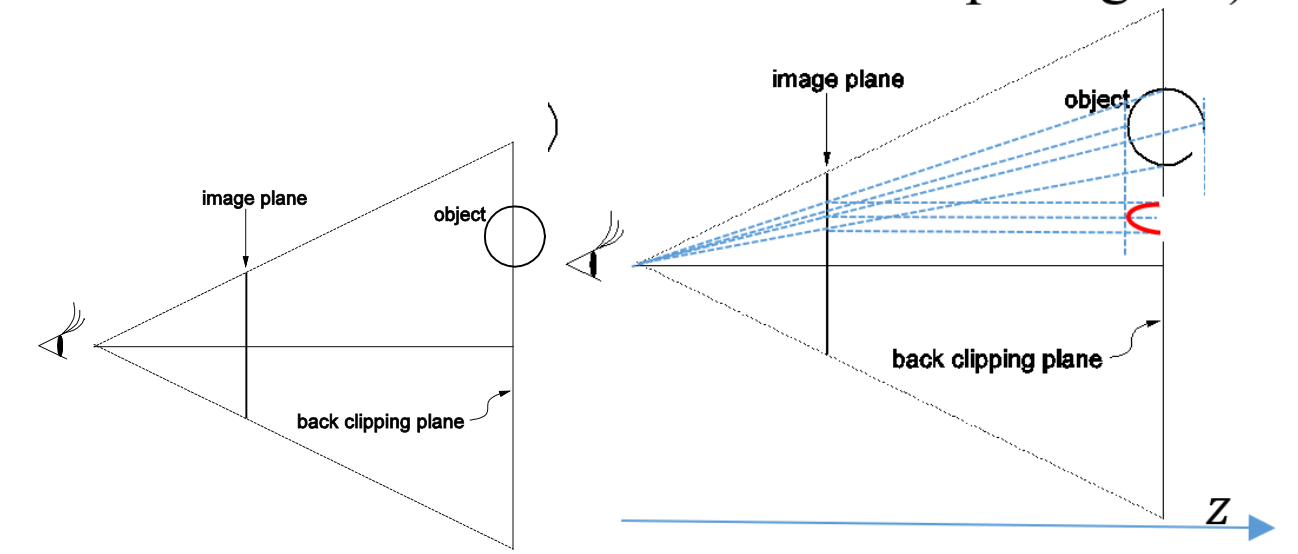
There are 6 borders (i.e., the 6 faces of the 3D cube), top, bottom, left, right, front,

back, 27 sub-regions, and thus the 6-bit code-words should be defined. Define the

trivial accept and trivial reject criteria

2. Show the output of the following object after perspective **transformation** and

clipping. (Indicate how you obtain the output figure.)



Perspective transformation is computed as:

𝑥’ = 𝑑 ∗ 𝑥 / 𝑧

𝑦’ = 𝑑 ∗ 𝑦 / 𝑧

𝑧’ = 𝑧

The key point of this question is that the z dimension will retain after the perspective **transformation**, which is different from perspective **projection**

3. A 3D object is rotated by 90 degrees about an axis passing from (1, 0, 1) to (1, 2, 1). It is then uniformly scaled 4 times relative to the origin. Write in their proper order the individual matrices composing this transformation.

