CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

CMT1[illegible]ual Computing

V.2 Ray Tracing

Xianfang Sun

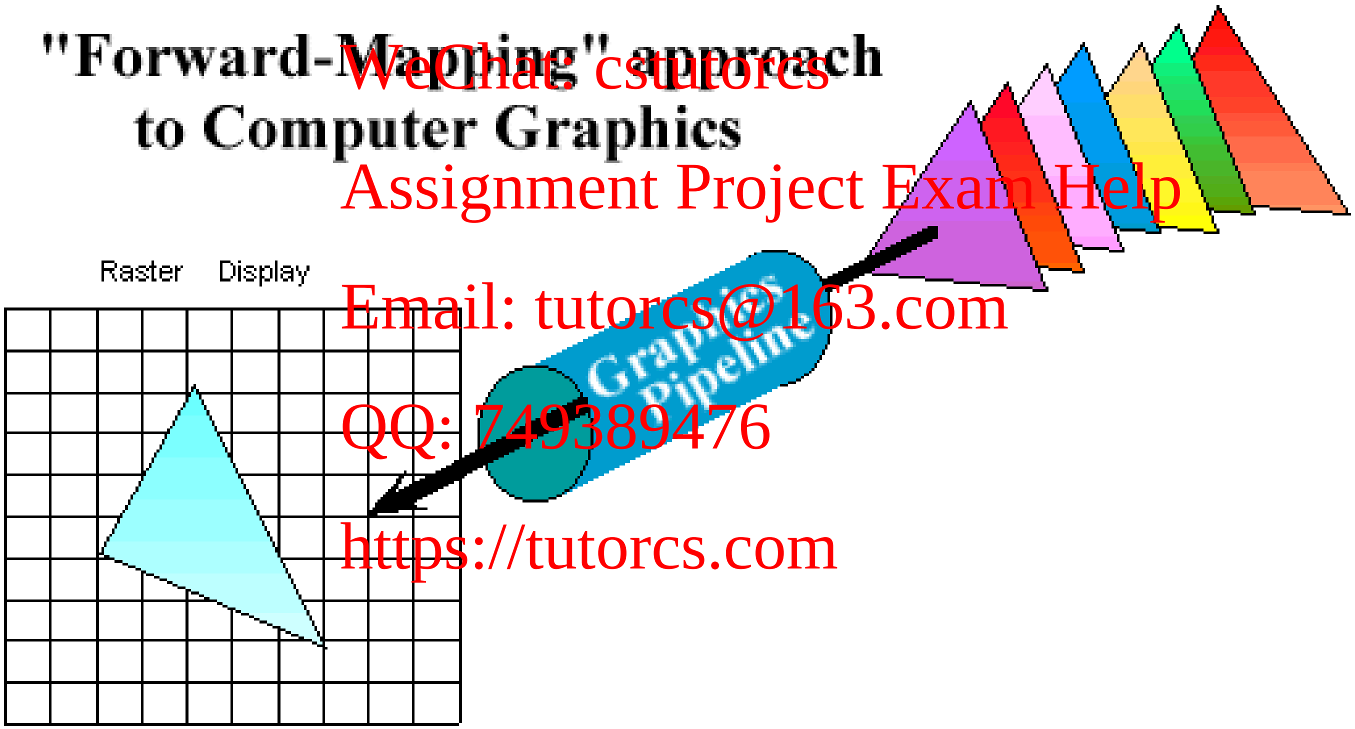
School of Computer Science & Informatics
Cardiff University

Overview

- Ray casting
- Ray tracing

Graphics Pipeline Review

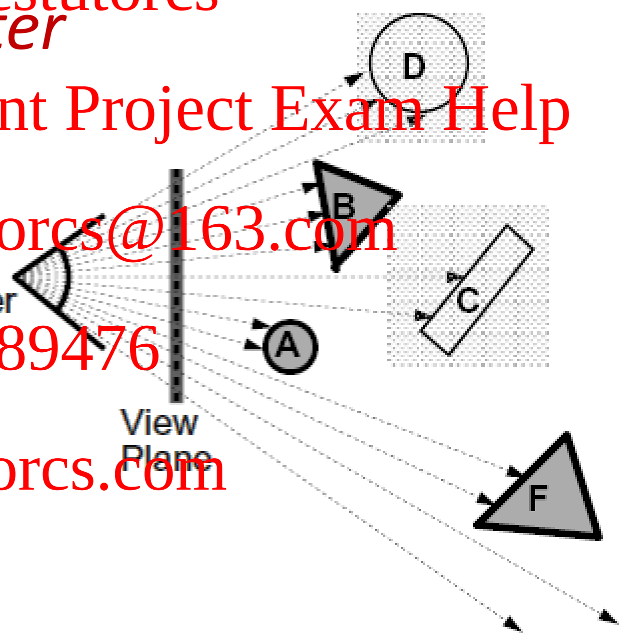
- Properties of the graphics pipeline
  - Primitives are processed *one at a time* (in sequence)
  - All analytic proc one *early on*
  - Scan conversion sation) occurs *last*
  - Minimal state re *immediate mode* rendering)



Ray Casting

- An alternative to pipeline approach: *ray casting*
  - Search along lines of sight (*rays*) for visible primitive(s)
- Properties:
  - Go through *all p*[illegible]*s* at *each pixel*

    (must have all pri[illegible] a display list)
  - Sample (rasterisation) *first*
  - Do analytic processing *later*
- *Inverse mapping* approach

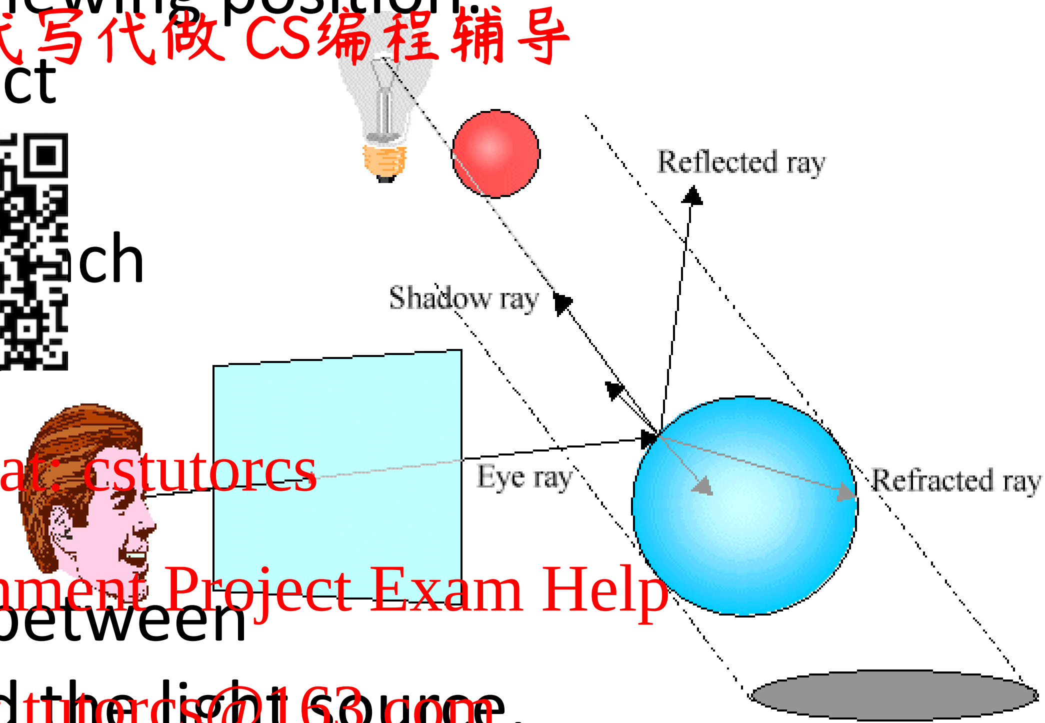


Global Illumination

- Ray casting properties:
  - Takes no advantage of *screen* space coherence
  - Requires *costly* [illegible] *computation*
  - Forces *per pixel* [illegible]*ation* evaluations
  - Not suited for i[illegible]te mode rendering
- In 1980 T. Whitted introduced *recursive ray casting (ray tracing)* to address global illumination

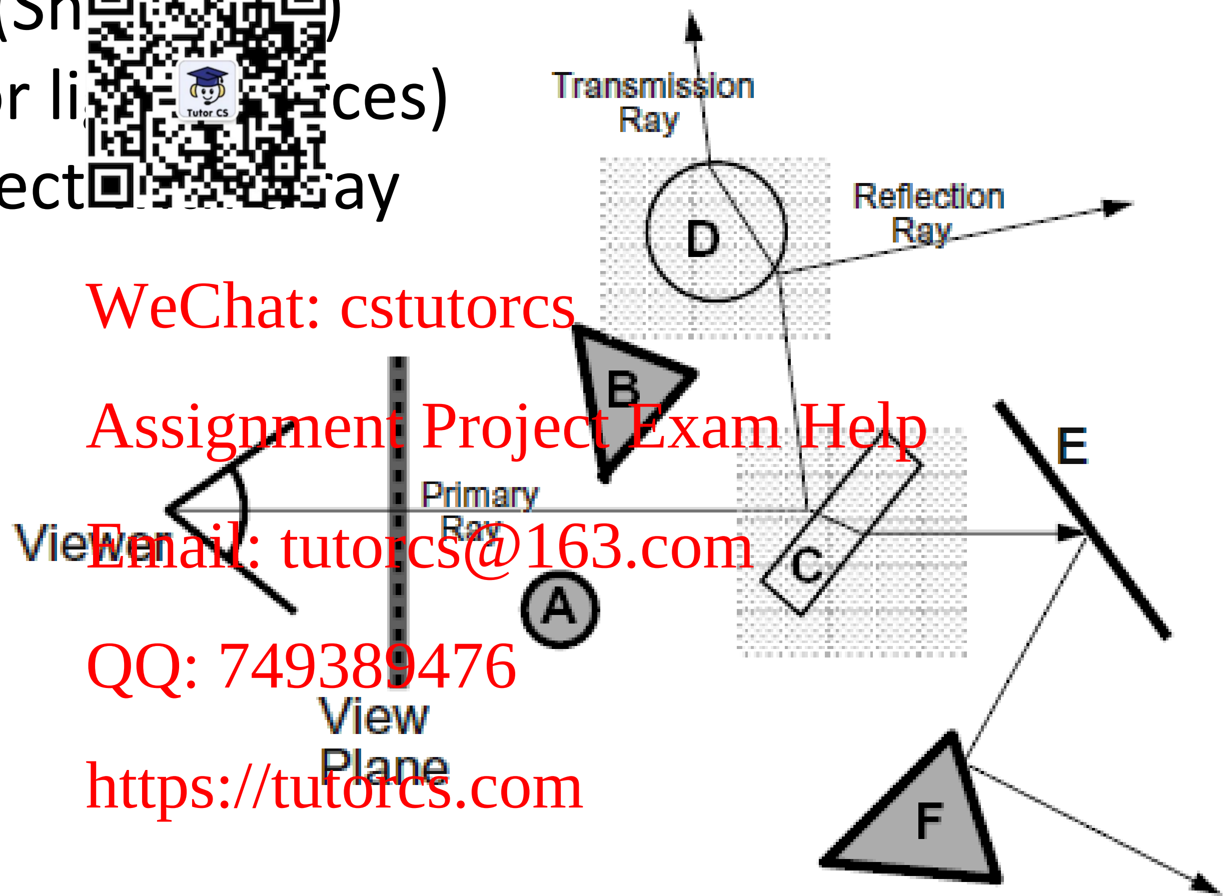
Ray Tracing

- For each ray from the viewing position:
  - Compute *visible* object along the ray
  - Compute *visibil*[illegible]ch light source fro[illegible] visible surface point using a new ray
  - If there is an object between the surface point and the light source, *ignore* the light source; otherwise, Phong illumination model is used to evaluate the light intensity
  - Can easily add reflection and refraction, etc.



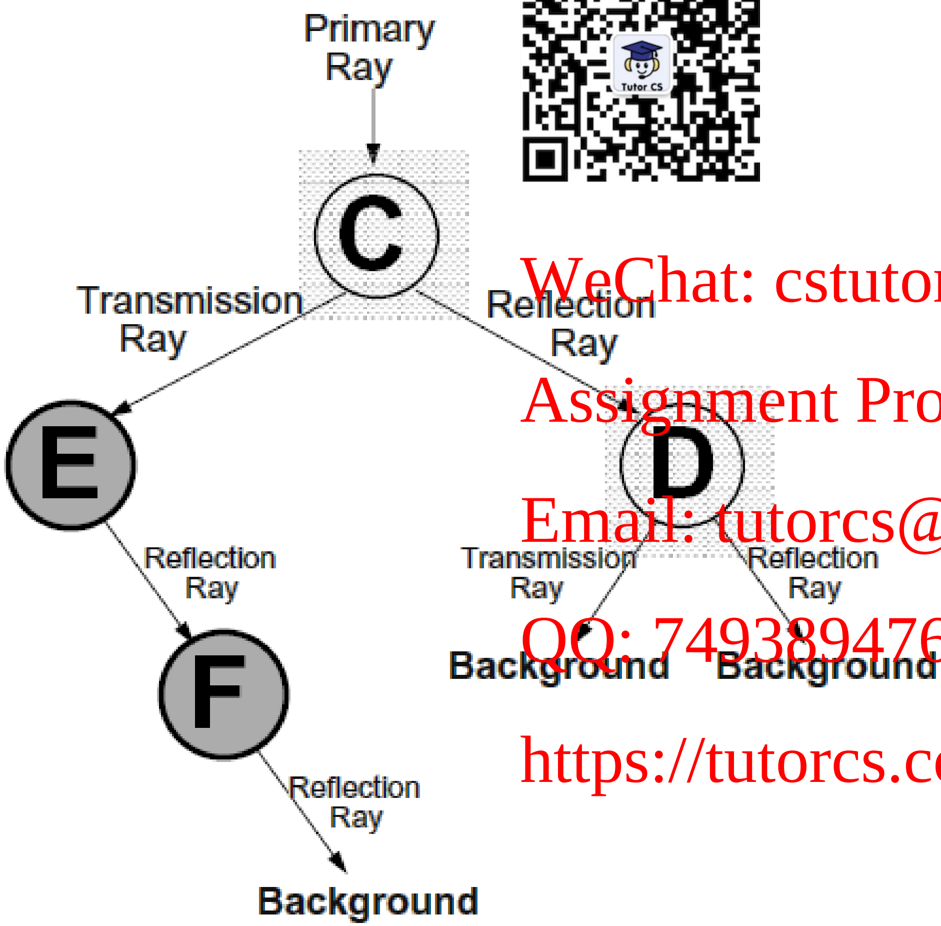
Ray Tracing

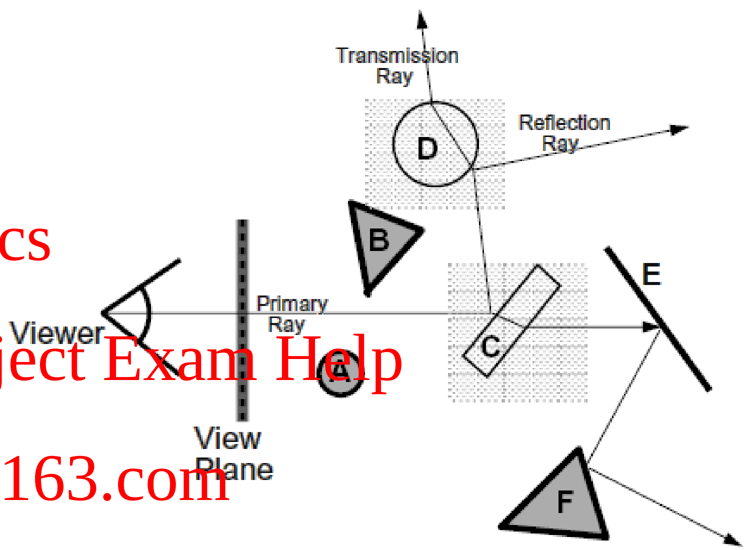
- For each object we need to know how to
  - *reflect* light (Phong's illumination model)
  - *refract* light (Sn[illegible])
  - *emit* light (for li[illegible]ces)
  - *intersect* object[illegible]ay



Ray Tracing Tree

- Move up backwards in tree and combine intensities as determined by *Phong's illumination model*



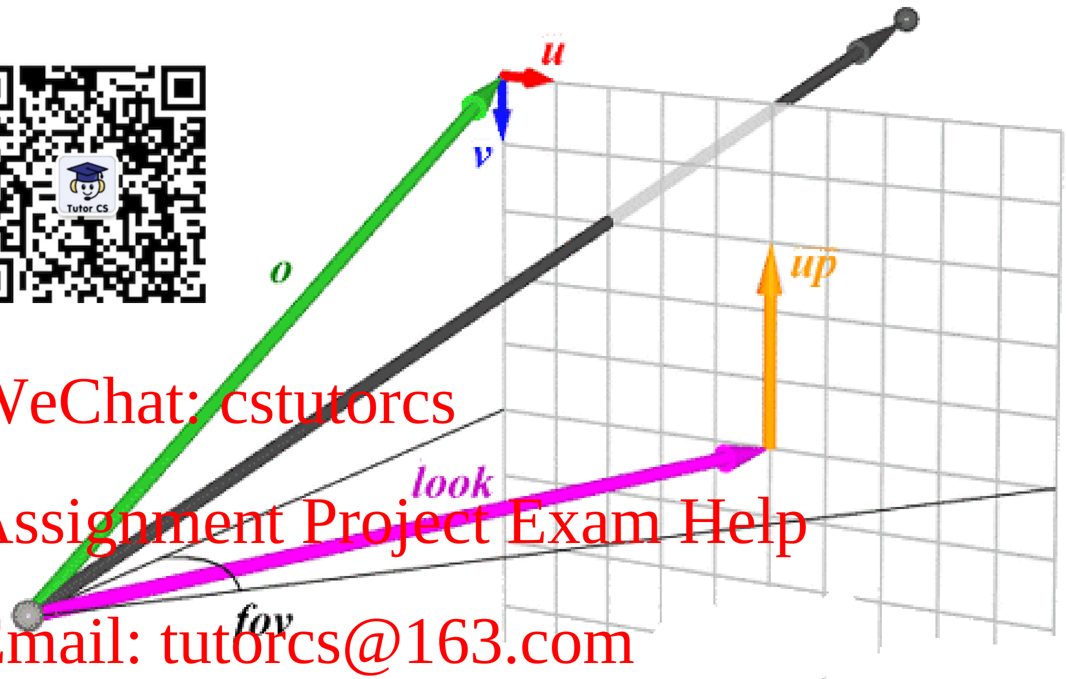


From Pixels to Rays

- Compute ray direction $v(x, y)$ for raster coordinates (x, y)

- $u = \dfrac{\text{look} \times \text{up}}{\|\text{look} \times \text{up}\|}$

- $v = \dfrac{\text{look} \times u}{\|\text{look} \times u\|}$



- $o = \dfrac{\text{look}}{\|\text{look}\|} \dfrac{\text{width}}{2\tan\left(\frac{\text{fov}}{2}\right)} - \dfrac{\text{width}}{2} u - \dfrac{\text{height}}{2} v$

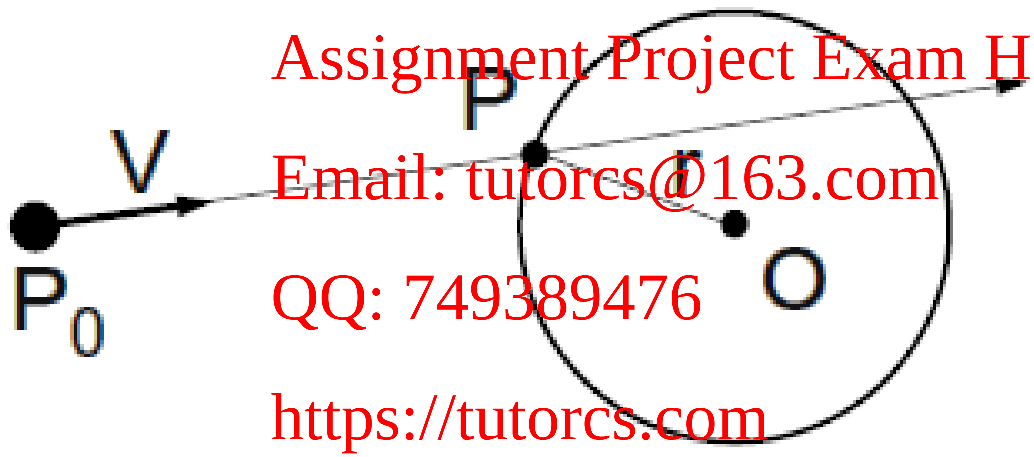
- $v(x, y) = (x u_x + y v_x + o_x; x u_y + y v_y + o_y; x u_z + y v_z + o_z)$

Ray-Plane/Polygon Intersection

- *plane-line intersection*
  - Ray: $P = P_0 + tV$
  - Plane: $P^T N + D$
  - Substitute: $(P_0 +$ … $D = 0$
  - Solution: $t = -\,($ … $) / (V^T N)$
- For intersection with polygon, check if intersection point lies inside polygon

Ray-Sphere Intersection

- *Intersect* a sphere with the ray (*algebraic*)
  - Ray parameterisation: $P(t) = P_0 + tV$
  - Sphere equation[illegible]$||^2 - r^2 = 0$
  - Substitute: $||P_0 +$ [illegible] $||^2 - r^2 = 0$
  - Solve: $t^2 + 2V^T(P$[illegible] $(||P_0 - O||^2 - r^2) = 0$



Ray-Sphere Intersection

- *Intersect* a sphere with the ray (*geometric*)
  - $L = O - P_0$, $t_{ca} = L^T V$
  - if $t_{ca} < 0$, no inte
  - $d^2 = L^T L - t_{ca}^2$
  - if $d > r$, no inter
  - $t_{hc} = \sqrt{r^2 - d^2}$, $\rightarrow$ $t = t_{ca} - t_{hc}$ and $t_{ca} + t_{hc}$

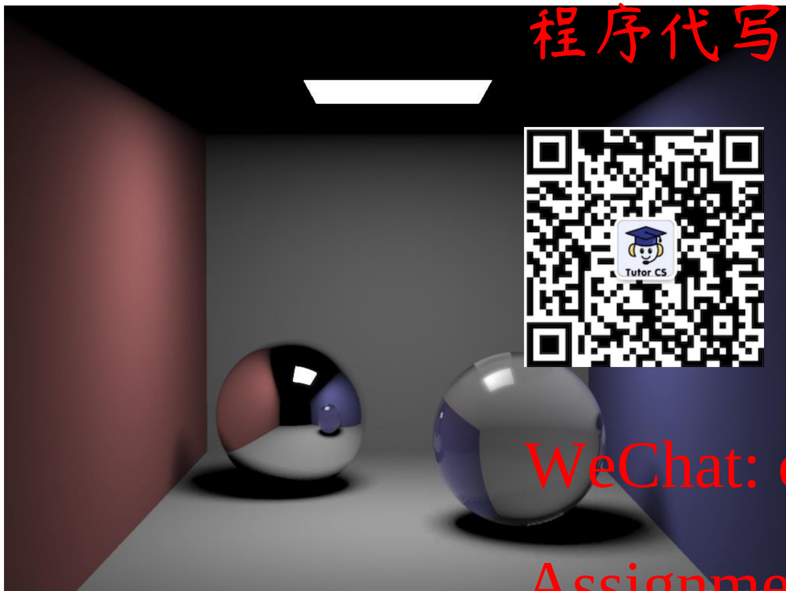
Ray Tracing Summary

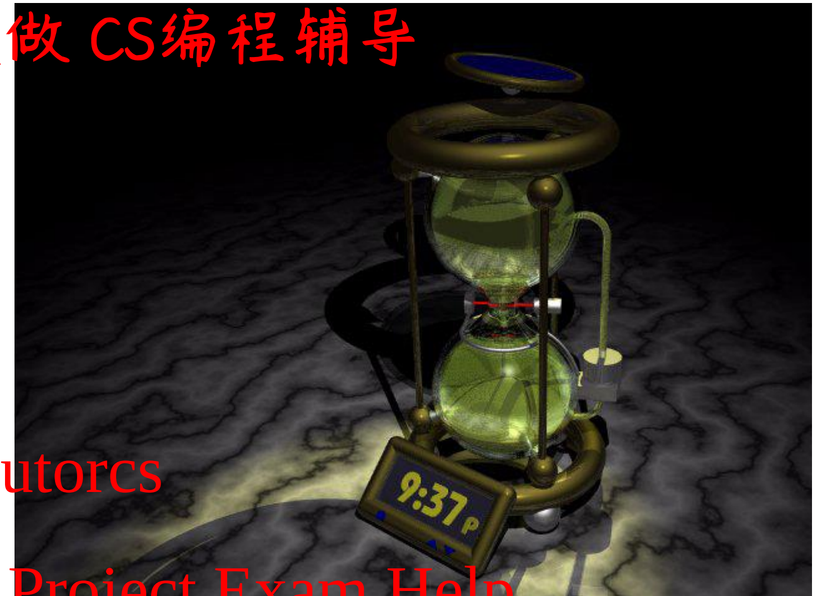
- Input: viewing position ***v***, look-at point ***a***, up vector ***u***
- For each pixel:
  - Create a ray ***l*** fr[illegible]iewing position ***v*** in direction ***d*** such that it pass[illegible]xel in the viewing plane
  - Set the colour t[illegible] return value of *raytrace(**v**, **d**)*
- Function *raytrace (**v**, **d**)*:
  - Initialise position ***t*** on ray ***l*** from ***v*** in direction ***d*** to infinity and the nearest object ***n*** to empty
  - For each object ***o*** in the scene
    - Compute intersection ***p*** of ***l*** and ***o*** closest to ***v***
    - If ***p*** exists and is closer to ***v*** than ***t***, set ***t*** to ***p*** and ***n*** to ***o***
  - If ***n*** is empty, return background colour, else …

Ray Tracing Summary (cont.)

- else …
  - If ***n*** is reflective and we haven't reached the maximum recursion depth level, compute perfect reflection vector ***r*** of ***d*** at ***t*** and call *raytrace**(t, r)*** to obtain reflected colour c_r
  - If ***n*** is transparent and we haven't reached the maximum recursion depth level, compute refraction vector ***r'*** of ***d*** at ***t*** and call *raytrace**(t, r')*** to obtain refracted colour c_t
  - For each light source k=1, …, m at position $\boldsymbol{l}_k$, cast ray from ***t*** to $\boldsymbol{l}_k$. If this line segment intersects with any of the other objects, ***t*** is in the shadow of this object. Otherwise compute the amount of light c_k reaching ***t*** *from k*
  - Return combination of colours c_r, c_t *and* c_k, k=1, …, m

Examples



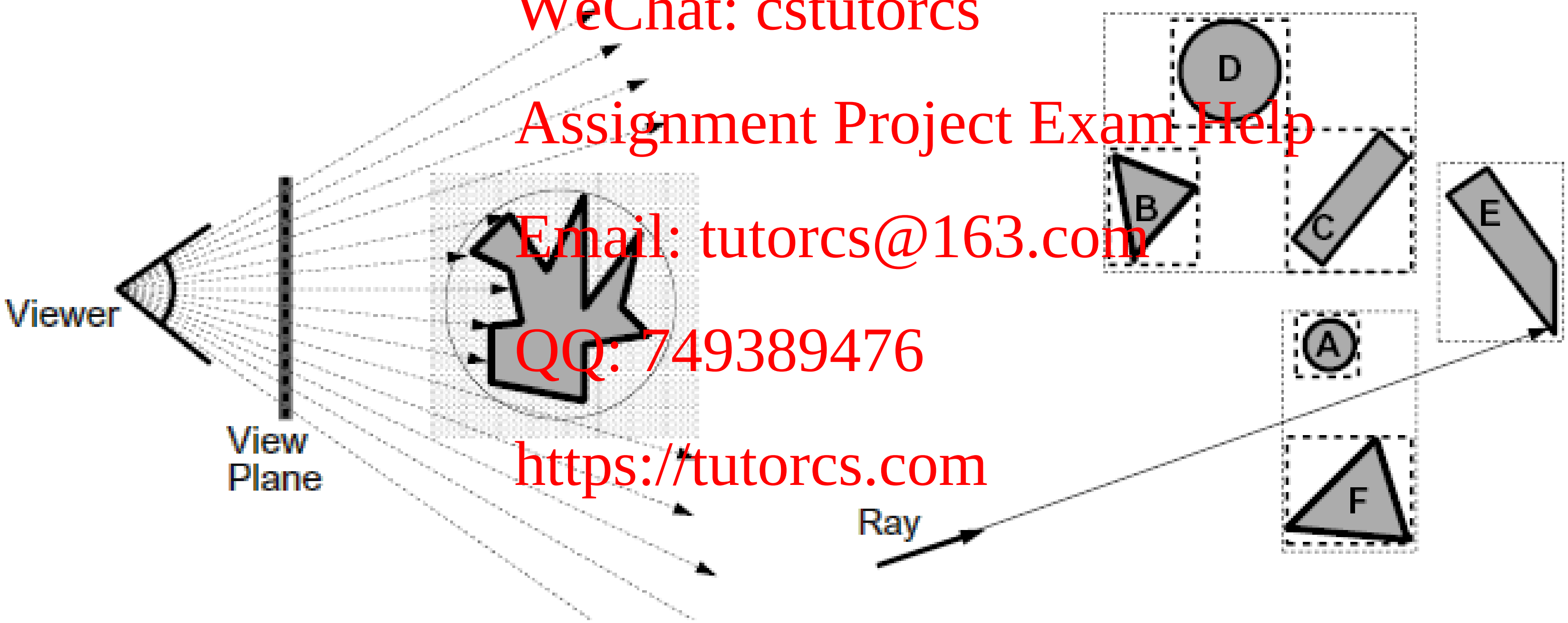


Properties of Ray Tracing

- *Advantages*
  - Improved realism (shadows, reflections, transparency)
  - Higher level ren rimitives
  - Very simple desi
- *Disadvantages*
  - Very slow per pixel calculations
  - Only approximate global illumination (cannot follow all rays)
  - Hard to accelerate with hardware
- *Acceleration* approach
  - Try to reduce number of intersection computations

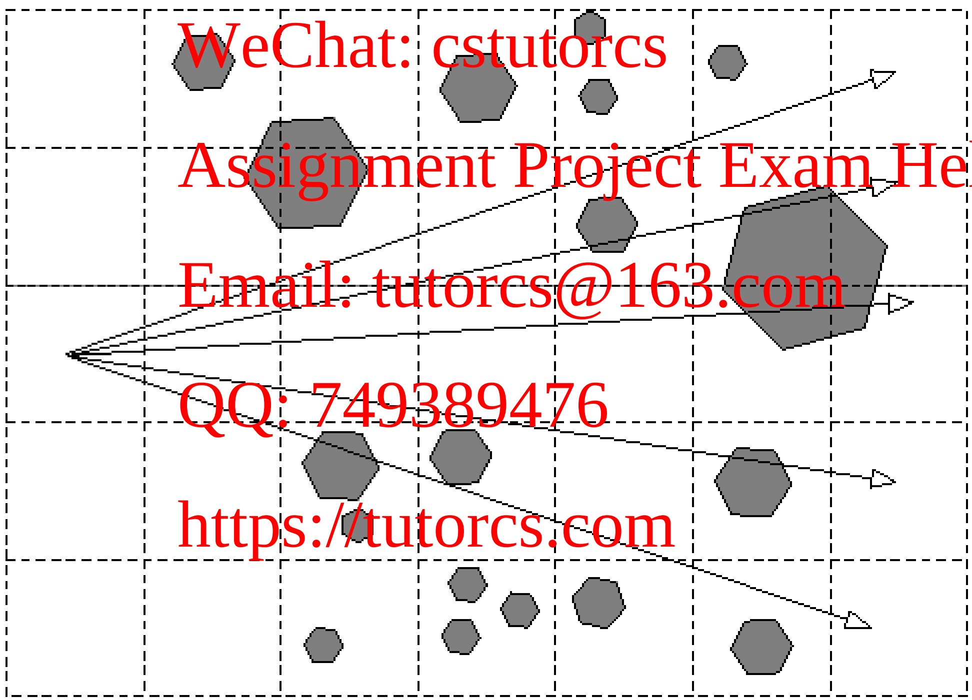
Ray Tracing Acceleration

- *Bounding volumes*
  - Check simple bounding volume for ray/surface intersections before checking complex shapes
- *Bounding volume hierarchies*
  - Construct and check hierarchical bounding volumes



Spatial Data Structures

- Create a data structure aware of the spatial relations
  - *Partition* space and place objects within subregions
  - *Only* consider subregions that the ray passes through
  - *Avoid computing intersections twice* if object lies inside multiple subregions



BSP Trees in Ray Tracing

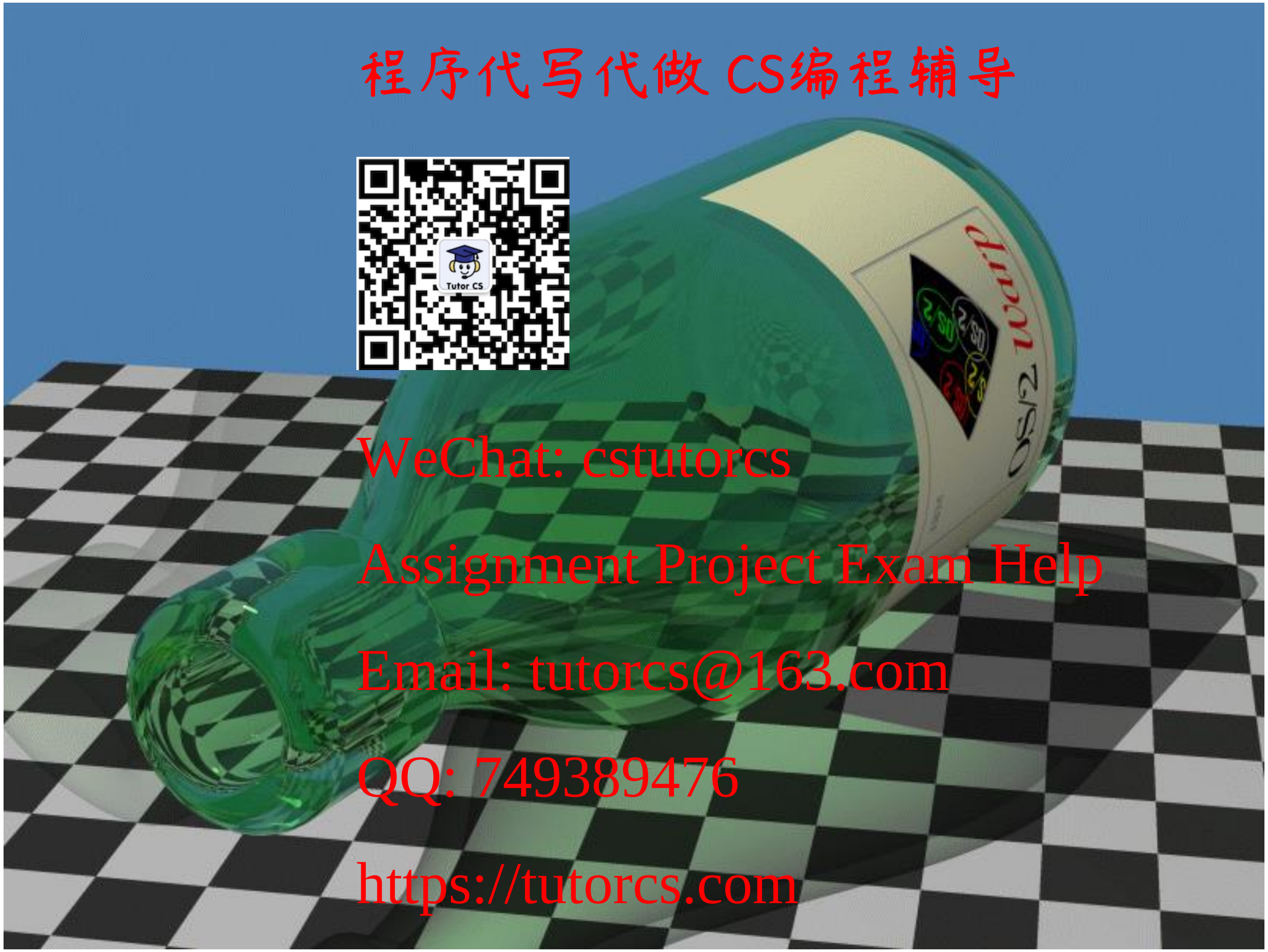
- Partition space using *BSP Tree*

Rendered Examples

Advanced Phenomena

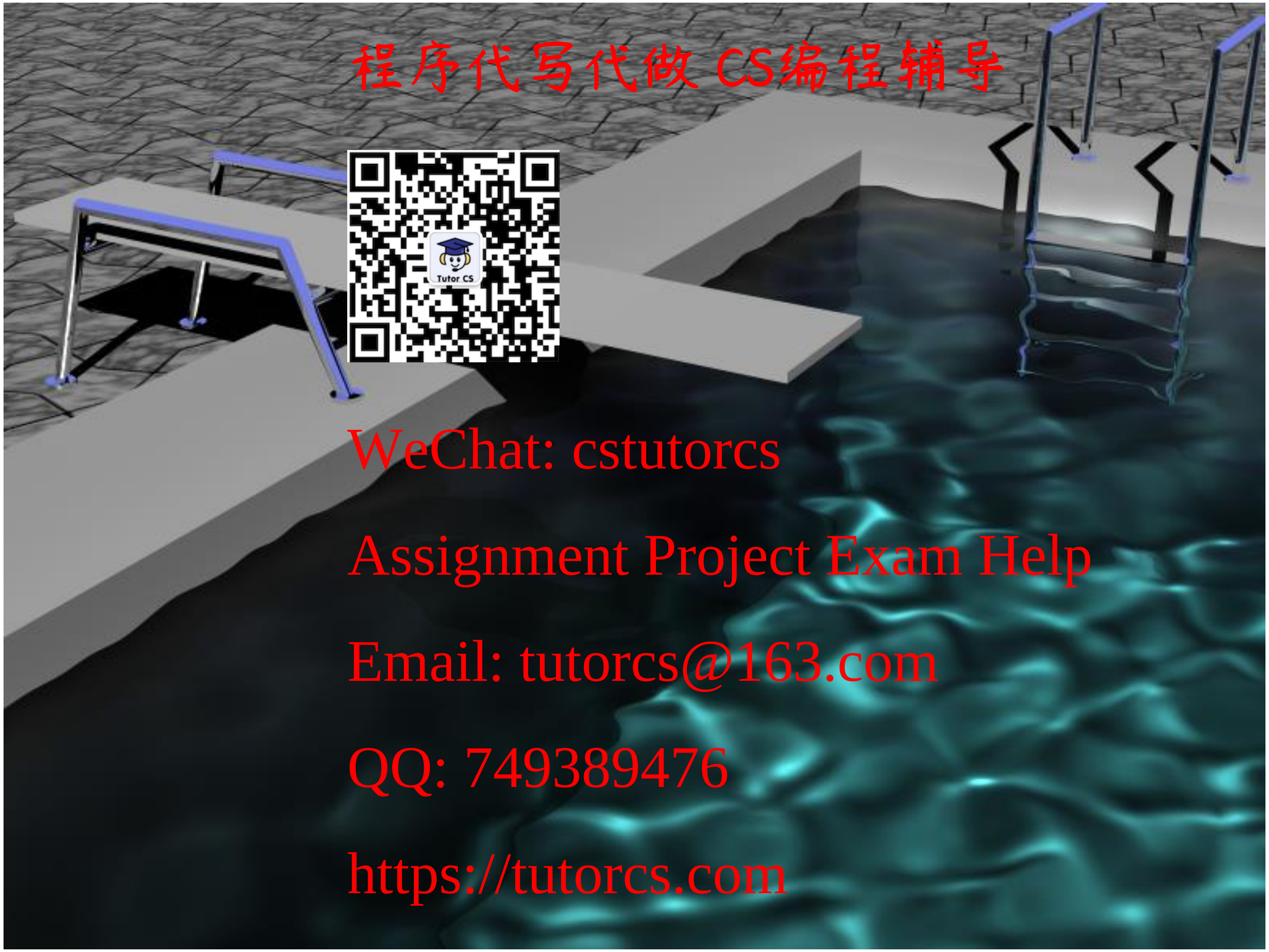
- Ray tracers can simulate (not always efficiently)
  - Soft shadows
  - Fog
  - Frequency depe ght
    (Snell's law is diffe ifferent wave-lengths)
- But can barely handle diffuse/ambient lighting
  - *Radiosity* is a global illumination scheme complementing ray-tracing for diffuse/ambient lighting

Examples



(by Oliver Kreylos, http://graphics.cs.ucdavis.edu/~okreylos/Private/RaytracingCorner/)

Examples

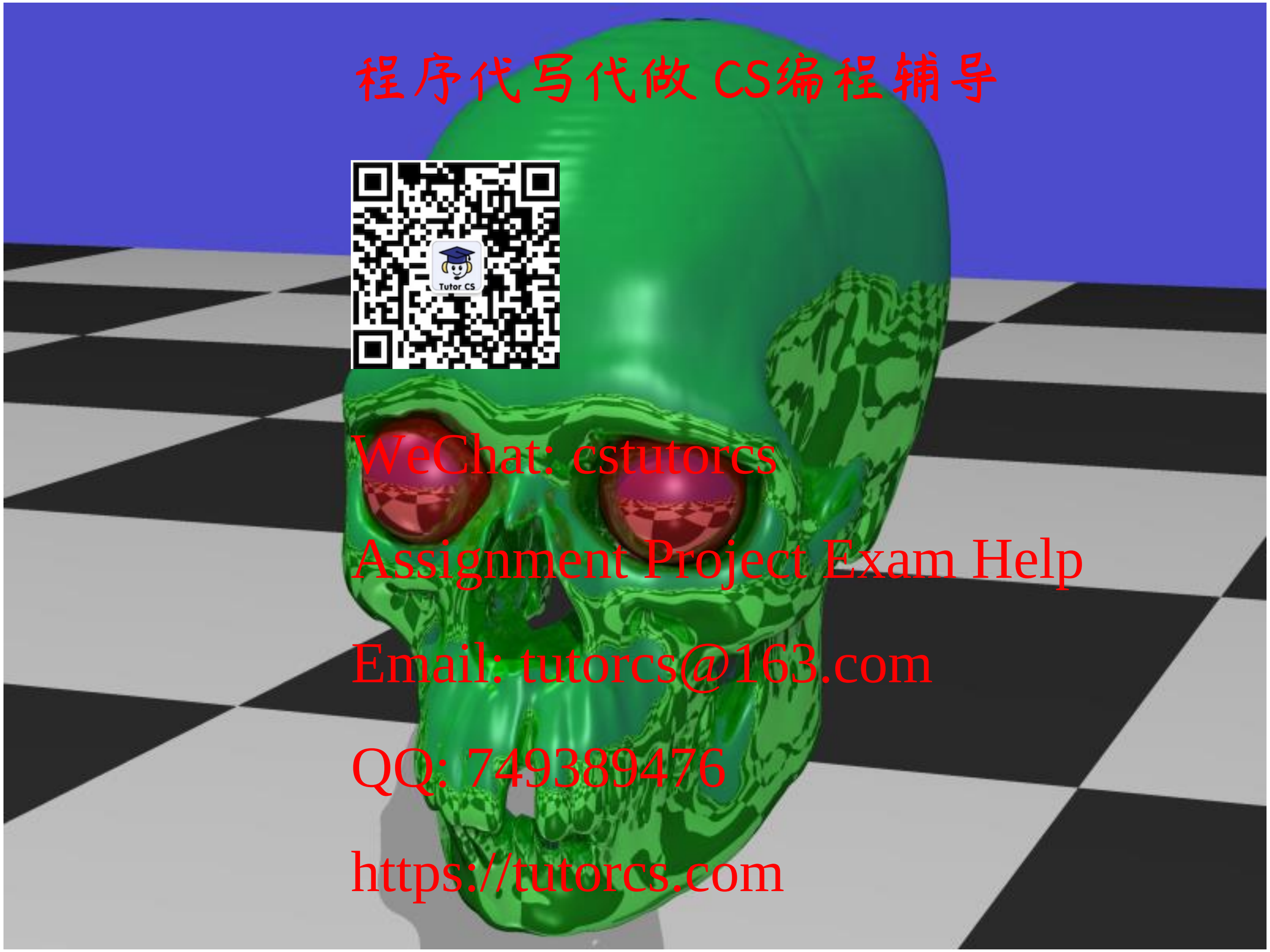


(by Oliver Kreylos, http://graphics.cs.ucdavis.edu/~okreylos/Private/RaytracingCorner/)

Examples

(by Oliver Kreylos, http://graphics.cs.ucdavis.edu/~okreylos/Private/RaytracingCorner/)

Examples



(by Oliver Kreylos, http://graphics.cs.ucdavis.edu/~okreylos/Private/RaytracingCorner/)

Summary

- What is ray casting? What are its advantages and disadvantages?
- What is ray tracing? What are its advantages and disadvantages?
- How can we compute the rays through raster points for ray tracing? How can we compute the intersections of such a ray and a plane or a sphere? How is this done for other shapes?
- How can ray tracing be accelerated?