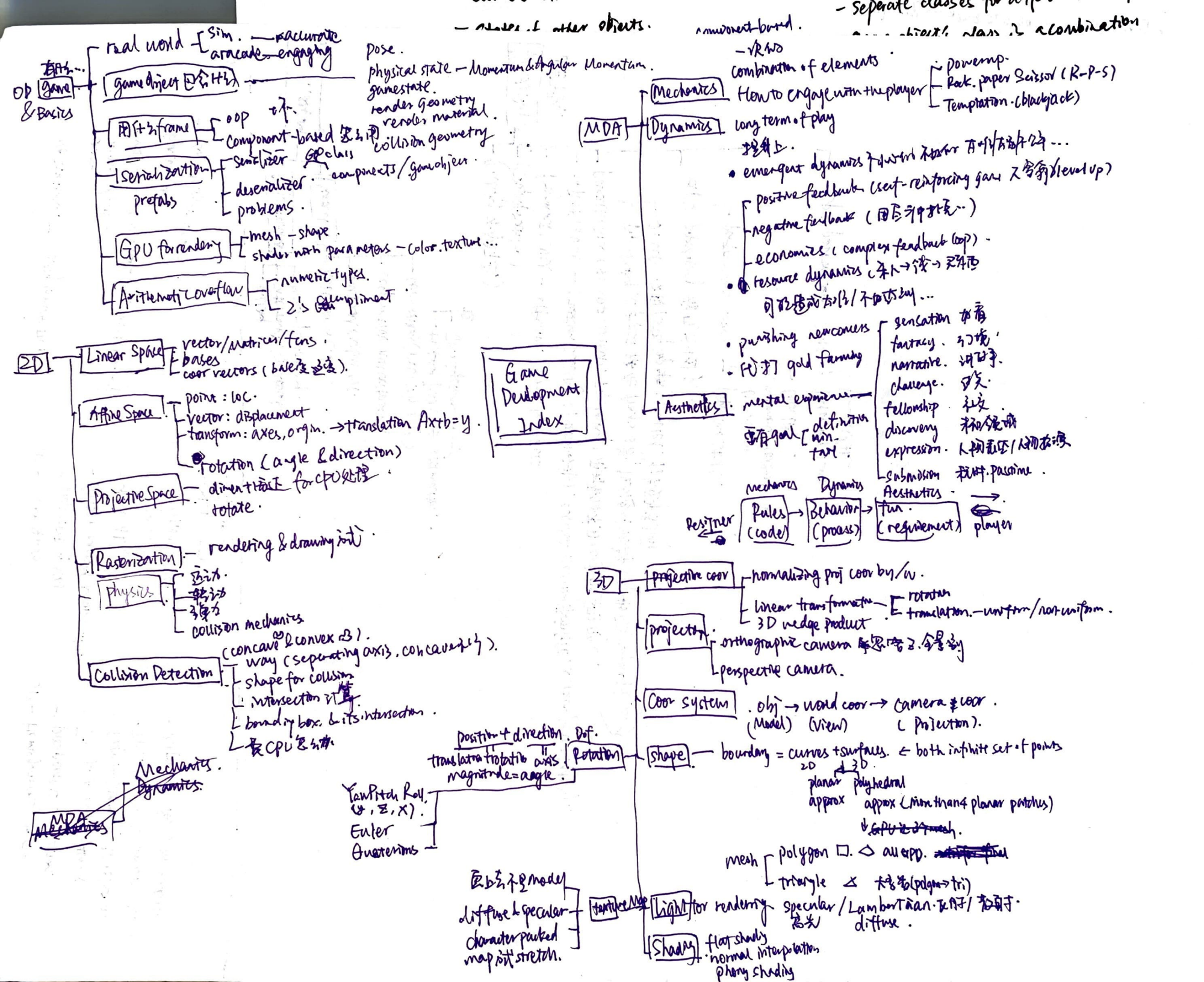

Game Development Index

vars to represent the state of the simulated world obj.

0D · architecture. – Unity Architecture.
- ~~Unity~~
- Input devices
- message, event handling, & multitasking
- scalar math
- game design & MDA

2D – linear spaces
- projective spaces
- sprite rendering & animation.
- game AI.
- physics.
- collision detection.
- Networking
- Audio.

3D – Homogeneous coor.
- prospective projection.
- Triangle based rendering
- Representation of 3D rotation.
- Transforms & Kinematic chains
- Performance issues in rendering

Multitasking
- **Asychronous**
  - polling
  - interrupts/callback.
  - message passing.
- **Parallel**
  - Aschronous calls.
  - mutual exclusion.
  - Cooperative multitasking.
  - coroutines (in Unity!!!)
    - ticking FSM. → starvation.
    - job system ←

Rendering 画与不画.
- (Culling 不可见不画)
- (Simplification.)
- fragment level – offscreen, normal 反, z坐标,
- object level – frustum culling (frustum之外), occulsion culling 被遮挡, screen graph culling.
- parallelism (SIMD等, MIMD, pipelining) 有对应的不同 rendering structure.
  - pipelining. (~~...~~ 排在 frags 之前, 但会造成任务不等待形成queue问题
  - SIMD 分配1 step 都 ~~...~~ parallel. 任务更平等分配.
- GPU pipelining 看笔记图 P14.
  - performance issue – pipeline stalls.
    - Overdraw.
    - memory contention.
    - locality.
- CPU performance issue – Batching issue.
  - bandwidth.

09/23 game.

- Data object that represent world objects.
  - state: vars to represent the state of the simulated world obj.
  - update: methods to compute new state from.
    - current state
    - states of other objects.
    - input from user.
- Mechanism for handling obj interactions.
  - collisions
  - attraction
  - attacks.

```
[ loop forever {
    clear screen
    draw
    update.
    handle interactions ]
```

Distortion.
- easier.
- simplified
- more real seeing.

- sim game
  - accurate. Simulation.
- arcade game (less accurate & fun, more engaging (driving...))
  - distorted simulation but

game object contains:
- pose info. transform matrix of where
- physics state. Momentum. Angular momentum.
- gameplay/AI state info.
- render geometry. represented as a set of polygons, shape be drawn on screen
- render material. - color & texture of the object, specified by shader, texture maps, color info, etc.
- collision geometry. Simplified geometry used for contact btw objects

OOP framework based.
- separate classes for different kinds of data
- game object's class is a combination of relevant parent classes.

[class Monster : GameObjectBase, Renderable, Collidable, PhysicsObject]

Then initialize the fields of the different classes

Problems
- require multiple inheritance (doesn't work well in C#.)
- multiple inheritance → issue.
  (both have "enable", 公共方法, name collision)
- explosion of classes.
  - different class for every combination of parent classes → a different class for each object

component-based.
- one game object class.
- a list of components → one field.
- have name, tag, childobj.

[monster or player?
test if they have some.

[use initializing with code.
Start( ).
new( ).
initializing with files — Serialize, Deserialize( ).
FromFile( ).
ToFile( ).
Serialize( )]

→ Type testing. (figure out monster or player).
objects are implicitly typed by the component they contain. To test a monster, check if it contains monster component.
if (maybeM.GetComponent<MonsterAI>() != null)

(Serialization 的应用场景) store all info to file. do not need recompile, just edit file.
~~save file.~~
~~reading in & out → serialization~~

Serializer
- convert game object to a stream of bytes
- can be saved to a file or transmitted over a network.

Deserializer.
- convert stream of bytes back into game object.
- update files in an existing game object. (for network update).

- Level Editors.

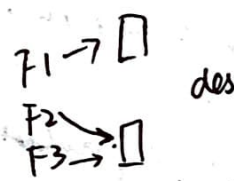
  - Compile with class definition from game.
  - Deserialize game objects from disk files.
  - Let user edit them.
  - Serialize game objects back to disk files

A basic Serializer.

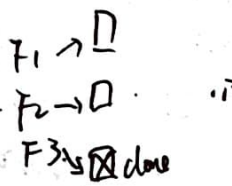
- Manually write serializer for primitive type.
- Every class have serialize method.
- To serialize an object recursively
- Deserialization is the same, but inverse.

problem ①.
Graph. Say we form a tree, & deep four for the tree.

DFS.



deserialize有问题.



if ← change

← do not change.

infinite version, fill hard disk.
or blow out stack.

problem 2. subclassing.

↳ a field filled with values of different types.

So:
- keep hash table of deserializer for different types.
- before writing object, write its type.
- when reading object.
  - first read type.
  - look up deserializer in the hash table.
  - call it.

keep track writing
- hash table. of objs you've written along with serial num.

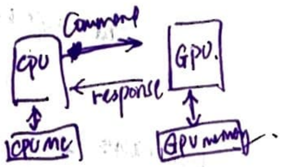
write.
- if in the table.
  write a magic number code.
  if not
    create magic code & serial number.
    make an & add to table.

Reading.
↳ read.
- keep hash table.
  read magic code.
- read obj.
  if "already"
    read ser num.
    look up in table.
  - if "new" obj code.
    read ser num.
    add to table.

Rendering.

draw all involves.

- A mesh: polygon to draw.
- shader. program run GPU to do drawing.
- a set of shader para.
  - coor transform.
  - color
  - texture.
  - lighting



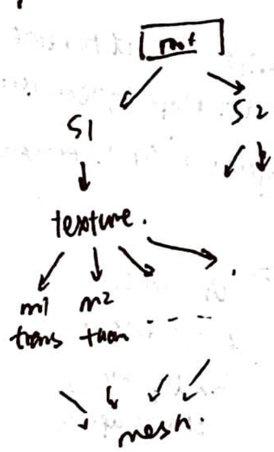
GPU:

- have its own seperate memory.
  frame buffer, shader, meshes, para blocks.
- fast.
- communicate btw CPU & GPU slow.
- minimize communication. & let them run independently in parallel

CPU command → GPU.
CPU ← response. GPU
memory memory.

Sort obj based on GPU.

DAG.



render.
push setting.
change for each node.
pop stack

change setting
if node is leaf.
send run.
else for each child.
render(child).
undo changes in settings.

- set texture
- run
- ~~undo set~~
- set.

→ with stack
push setting (about to changes).
change setting by node.
if. leaf.
run.
else for each render
pop the stack

start
~~potted~~ Deserialize.
call to startup
loop update/draw: quick & constantly.

polled update: ~~going~~ ticking when nothing happens.
event handler: when sth happens
virtual method. "hit me"
event queue. → hit you".
callbacks. : other things happens ~~60 fps=16ms per frame~~.

realtime animation. 60 fps = 16ms per frame.

- update routine. [illegible].
- physics.
- event handling.
- audio
- rendering

Multithreading
Can do multithread, would help if tasks can run independently without much interaction.

update [illegible].

- coroutines [illegible].
- seperate threads. update, render, physics [illegible]. 16ms/frame.
- Jobsystem. Queue of long-running operations. seperate one at a time.

path planning
  seperate thread.
  que of request → planner thread → send event back to game.

Game Obj - a collection of Component.
  UnityEngine.object
^
Subclass of   has transform - info of how appear on screen.

  - a list of Component & a list of child GameObjects

  methods:
    GameObject.Find(string)
    GameObject.FindWithTag(s)
    GameObject.FindGameObjectsWithTag(s)

  • instantiate.
  • Destroy.

Component.
  Combined with GameObject.
  mGO.getComponent<Transform>().position

Transform Component
  Coordinate change (position. rotation. scale)

GO hierarchy → scene graph.

Common Component: camera, Transform, renderer.
  Light, skybox, Collider, RigidBody

GetComponent<Type>()
  — . InChildren
  — InParent.
  — s.
  — s InChildren.

Making own components.
  new class MonoBehavior.
  add fields.
    • public.
      - serialized.
      - Editable.
  add message handler (Update)
  else.

Initialize Component.
  - avoid constructor ① 会在 deserialize 之后 call 会 ∴ 会 overwrite.
    fields private 会 …… ~~因会在 load thread~~.
    public 会 erases 之后 level file for data.
    ② 这在 load thread not main thread, can't call most Unity fns.
  - put initialization in Start() message handler, 有些 const code written 会 得到 结果不同

Attributes.
  C#.

Update & message handler.
  Reflection: ability to write code at runtime asks questions about the type system itself.
    X.getType(). → type object with fields, methods, objects.

  Common.
    Start: awake().
      Start().
    Update: Update().
      LateUpdate()
      FixedUpdate()
        physics update
    Collider messages.

Execution
  ~~Loading~~ level load.
  Create obj & deserialize data.
  Awake().
  Start().
  Caveat: own thread.
    ✓: let games running before loading.
    ✗: operations not allowed to perform from Awake().

Unity Serializer.
  ① refuse to serialize certain objs.
  ② 不支持 subclassing.
  ③. ?!$@# → null.
  ④ cannot handle DAGs & cycles in certain cases

1.6-1.7. 7.1-7.5.

1006. game.

geometry:

points. lines, planes, Distances, Angles, parallelism. shape,

represent:

numbers. vectors, matrices, Dot products:

Linear spaces.

- a set together with Addition & scalar multiplication
  - commutativity
  - associativity
  - $0x=0$
  - $t(x+y)=tx+ty$.
  - $(x+1-1)x=0$.

Linear fcns

- Functions that commute with addition & scalar.
- form linear spaces.

Coordinate vector & Bases.

Bases.

Set of vectors multiply by different scalar & add result.

minimal!

not unique. infinite amber of bases.

2D - 2 vectors. 3D, —> 3 vectors.

focus on orthonormal bases

- unit=1
- mutually perpendicular.

$x=x_1b_1+x_2b_2\cdots+x_nb_n$

$y=y_1b_1+y_2b_2+\cdots xy_nb_n$

weight $x=(x_1\cdots x_n)$, $y=(y_1\cdots y_n)$ —> coordinates.

change bases, coord changes.

mapping from V to R^n (space of coor vectors)

Coor representation:

* Let $f: V \to \mathbb{R}^n$ be fcn mapping elements of the linear space to coor vectors. representation.

  f is a linear fcn.

* . f from.

physical space ≠ linear space.

- position, Distance, Angle ... ← do have.
- Addition, Multiplication, "zero" ← do not have.

Affine space.

- a set of points,
- a seperate set of vectors represent displacement.
- a set of transformation. for parallelism of lines

point - location. $P+P$ ✱ $P+V=P$. scalar * P X.

vector - displacement.

origin "zero" point

Transforming coor system.

- change of axes./ bases
- change of origin.

Linear $y=ax$.

affine $y=ax+b$.

change of object.

Vector : change of basis matrix.

points : plus a offset.

Angles.:

distance distance around circle.

inner/~~out~~ exterior product.

rad : $0 \sim 2\pi$.

Inner

$u \cdot v = |u||v|\cos\theta$: 夹角 θ.

$u \cdot v = u_x V_x + u_y V_y = |u||v|\cos\theta$.

or just transfer to unit vector then dot product = $\cos\theta$.

Angle as area.

How area created: one sweep along the other

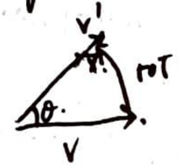
Area of their intersection normalized by r of circle.

$u \wedge v = u_1 v_2 - u_2 v_1$. $\begin{bmatrix} u_1 & u_2 \\ v_1 & v_2 \end{bmatrix}$ (det)

$u \wedge v = 0 \rightarrow u \parallel v$

max when $u \perp v$ (exterior product).

Rotating the x-axis. $v' = (\cos\theta, \sin\theta)$.



y-axis.

$v' = (-\sin\theta, \cos\theta)$.

(x, y).

$R_\theta = (x(1,0) + y(0,1))$.

3D:

$v' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} v$.

(万向节) discontinuity $2\pi \rightarrow 0$.

~~Vectors~~

compare

Representing Orientation

- Orientation is how an obj is rotated.
  - direction as vectors: many vectors represent the same dir. redundant, a 2 vector → 1 dof
  - : continuity.

~~Representing direction~~

projective spaces. — computer graphics represent things in terms of this

Let in n dimension

- it's a set of lines through origin in R^{n+1}.
- Vectors from R^{n+1}. 不等于 0.
- Idea: add a redundant dimension to make things easier.

projective coordinate.

origin (wx, w), $w \neq 0$.

若 for $(x, 1)$, translate with amount t (add t to 1st)

$\begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} (x, 1)^T = (x+t, 1)^T$.

$\Rightarrow \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} (wx, w)^T = (w(x+t), w)^T$.

2D: (x, y).

$\Rightarrow (wx, wy, w)$.

translation matrix $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$

~~Projective direction~~

* Coor transform. in 2D. $(x, y, 1)$ (wx, wy, w).

$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$ ⇒ 3D同理

Hierarchy.

$v_i' = W(P(S(U(F(H\,v_i)))))$

$v' = (WPSUFH)\,v'$. ← matrix associative

Transform properties.

- forward
- up
- right

2D → 3D

$(x, y) \rightarrow (x, y, 0)$.

3D → 4D.

$(x, y, z) \rightarrow (x, y, z, 1)$.

2018 cs game · Rasterization ~~material color~~ → ~~seperate systems faster in GPU~~
geometry where →

physics simulation → realistic motion of objects. – collision / gravity.

Models of physics.

Human Scale.
- Aristotelian
- Newtonian.

Aristotelian:

~~Newton~~ object have natural place, naturalization.

eg. earth/water on land.
Fire/air: in the sky.
Aether: celestial objects.

👁 fall & stop.
👁 for daily life

Newtonian:

one kinds matter with one natural state.

constant speed & direction

change direction and/or speed. → disturbance.

if disturbance ends, continue their last direction & speed.

👁 more until interrupt

👁 precise quantitative model

Newtonian of point particles

* (think of) objects → infinite small points. (no rotation/size, just position)

* keep a straight line move at constant speed

* State
  - position
  - velocity $r+tv$.

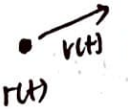
* apply force.

* Force – rate of change of velocity 改变速度

$$\frac{dv}{dt} = F.$$

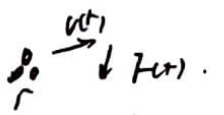
$$v(t) = v(0) + \int_0^t F(t)\,dt$$

$$r(t) = r(0) + \int_0^t v(t)\,dt.$$



→ 45.

* Disturbing.
  - effect of force is inversly proportional to the amount of stuff it's acting on
  - the amount mass - m.



Thus. $\frac{dv}{dt} = \frac{F}{m}$. $(F = ma)$.

$$v(t) = v(0) + \int_0^t \frac{F(t)}{m}\,dt$$

$$r(t) = r(0) + \int_0^t v(t)\,dt$$

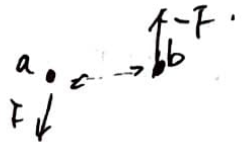
Linear Momentum.

(forces comes in pairs).

- Forces due to interaction btw two or more.

thus, interactions are reciprocal 상호작용

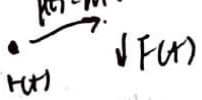
thus, net force = 0



rate of change.

$$\frac{d}{dt}(V_a(t) + V_b(t)) = \frac{d}{dt}V_a(t) + \frac{d}{dt}V_b(t) = F + (-F) = 0.$$

Momentum



$p(t) = m\,v(t)$.

$$p(t) = p(0) + \int_0^t F(t)\,dt.$$

$$r(t) = r(0) + \int_0^t \frac{p(t)}{m}\,dt.$$

$$\frac{d}{dt}(p_{(a)}(t) + p_b(t)) = F + (-F) = 0.$$

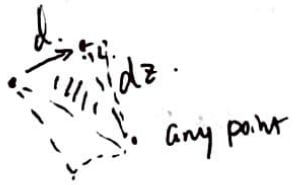
total momentum 불변. 내부작용 없

$$\frac{d}{dt}\sum_{i=0}^{i=0} p_i(t) = 0.$$

Angular Momentum.

- inertial motion. i.e no force apply

compute area :→ half of 평행사변형 $d \cdot dz$.



~~distance traveled~~.

10/4. Polygon Rendering

Rasterization.

line & triangles 의
- material. : color/what.
- geometry. : where.

projection : world coor → pixel coor (translation, rotation, scaling 등 for map)

ScreenCoor $S = Mw$ ← world coor.

matrices (screen coor)

projection matrix.

Linear Interpolation (lerp).

a & b two points in linear space.

lerp (a,b,s) = a + s(b-a).

s: 0 ~ 1.

lerp(a,b,0) = a

lerp(a,b,1) = b

lerp(a,b,0.5) = midpoint of a & b.

can have scalar

linear fcn. a_1, a_2 ... ⇒ lerp(a_1...) + lerp(a_2...).

lerp(f(a), b, s) = f(lerp(a,b,s))

lerp(a, f(b), s) = f(lerp(a,b,s)).

Linear Rasterization

DrawLine(w_1, w_2)

for each t from 0 ~ 1 (going in some small steps).

w = lerp(w_1, w_2, t)

s = Mw.

set color at s.

faster ⇒.

DrawLine(w_1, w_2).

$s_1 = M(w_1)$

$s_2 = M(w_2)$

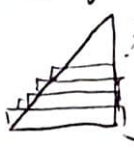
for each.

s = lerp(s_1, s_2, t).

set s.

Bresenham Algo. 생략하지만 찾아보기.

Rasterizing Triangle.



→ 좌우

DrawFlatTri(a,b,c)

$\Delta y = a_y - b_y$

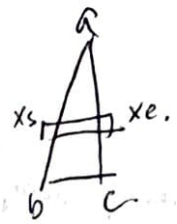
for t = 0 ~ Δy ← scanline 개수.

y = b_y + t. b_y → bottom y

x_s = lerp(a_x, b_x, $\frac{t}{\Delta t}$)

x_e = lerp(a_x, c_x, $\frac{t}{\Delta t}$).

for pixel from (x_s, y) to (x_e, y)



non-flat

좌우 → scanline.

convex polygons ⇒ triangles.

quadrilateral

choose point inside → fan break up.

Materials & shaders.

└ shader & its parameter

- GPU subroutine.
- takes a set of coors & returns it color

2D a fixed color.

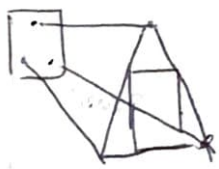
shader ignore its arguments.

texture map.

load raster image into memory.

- tag each triangle vertex with UV coor
- linearly interpolate the coor for pixels inside the triangle.
- assign each pixel the color from the image at interpolated coors ↓

idea: map each coor



Transparency

(RGB α).

α: 0 transparent

alpha blending 1 opaque.

$O_R = \alpha_1 I_R + (1-\alpha_1) O_R$.

O_G

O_B

Depth-ordering

blending..
sprite rendering

6. Conservation of energy

start & fall, the same KE.
inbtw, disappear
↓ increase.

In game:

alternatives are:

everything slow down & stops.

everyting explode.

only approximate:

- numerical error accumulates
- leak energy
- create it.

Electrostatic Force.

Like gravity. except

- mind boggling stronger. (gravity is patheric)
- determined by charge rather than mass.
- charge signed.

A, B two particle $r = B - A$.

$F_{AB} = -k_e \frac{q_A q_B}{\|r\|^3}$ $= \frac{q \cdot q}{} \; 9 \times 10^9$

$\Rightarrow$ about 10^{20} times stronger than gravity.

Collision Dynamics.

Atoms.

- negatively charged electrons.

positive charged nuclei.

attract one another to form atoms.

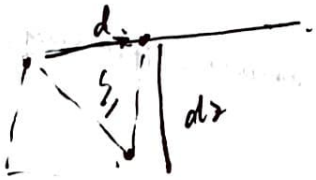
Quantum magic keep them from touching

Depth-ordering

How to another second.

She d & dz same,

the area the same.



* Sweep with constant rate.

* Va conserved, P conserved, area * mass conserved.

angular momentum: the rate area is swept.

area: $r \wedge V$.

area of tri: $\frac{1}{2}m(r \wedge V) = \frac{1}{2}(r \wedge p)$.

angular momentum = $L = m(r \wedge V) = (r \wedge p)$.

AM ⇒ Conserved.

Forces

gravity -- all mass attract one another.

- in proportion to the mass of both bodies
- inverse proportion to the square of the distance btw

$r = B - A$ (B, A are particles)

L → vector

gravatity force: $F_{AB} = Gr\frac{m_A m_B}{\|r\|^3}$ for some magical constant $G = 6.7 \times 10^{-12}$.

$F_{BA} = -F_{AB}$

terrestrial gravity.

Only case.

Throw a ball.

leave hand at height h & velocity V.

As it ~~falls~~ arises, gravity decelerate until rest.

the gravity accelerate it downwards.

Kinematic Energy.

- the same magnitude of velocity
- KE: Scalar.

mass * square of v

$= \frac{1}{2}mv^2$

a particle interact with an atom, it feels forces from both
- the electrons (repulsion)
the nucleus (attraction).

$$F = k_e q_e \sum_{p \in Atom} \frac{r_p q_p}{\|r_p\|^3}$$

Momentum & energy conserved.
thus we can compute ending v. directly from their starting v.
etc.

mass m_1 m_2
velo u_1 u_2. befor collision. v_1 v_2 after collision.

$$m_1u_1 + m_2u_2 = m_1v_1 + m_2v_2$$

$$m_1u_1^2 + m_2u_2^2 = m_1v_1^2 + m_2v_2^2$$

$$v_1 = \frac{u_1(m_1 - m_2) + 2m_2u_2}{m_1+m_2}$$

Same but reversed.

Contact Geometry
where they hit.
- Tangential.
- Normal.

$$n = \frac{x_1 - x_2}{\|x_1 - x_2\|} \qquad u_1 = n \cdot v_1 \qquad u_2 = n \cdot v_2$$

$$v_1 = \frac{u_1(m_1 - m_2) + 2m_2u_2}{m_1+m_2}$$

v_2 [illegible]

$$v_1' = v_1 + (v_1 - u_1)n$$

$$v_2' = v_2 + (v_2 - u_2)n$$

Other Forces.
Spring $F = -kx$.

$$\frac{dv}{dt} = \frac{-kx}{m}$$

dissipative forces.
Friction F_n $\rightarrow$ $F = -\mu\|F_n\|\frac{v_t}{\|v_t\|}$

Fluid Force.

$F \cong -cv$. for viscous damping 粘性阻滞. (low v)

or $F \cong -C\|v\|v$ for drag (high v)

Physics simulation.

- discrete time → finite timestep.
  - integrate.
  - solve constraints among bodies (collision & joints)

inaccuracy → instability

Euler integration.

~~easy & obvious to implement.~~

$\int dt = \sum \Delta t$.

$x_t = x_{t-1} + v_t \Delta t + o(\Delta t^2)$

$= x_0 + \sum_{t=0}^{T} v_i \Delta t + o(\Delta t^2)$ same as v.

loop

For each particles:

recompute net force F

$a = F/m$.

$v = v + a\Delta t$

$x = x + \Delta t \cdot v$.

[but error is quadratic] ← instable.

1D - $\frac{dx}{dt} = -kx$.

$x_t = x_{t-1} - k x_{t-1} \Delta t$

$= x_{t-1}(1 - k\Delta t)$

if k 太小 $1 - k\Delta t \approx < 1$.

$|x_t| < |1 * x_{t-1}|$

x converge to 0 ✓

(Spring 硬的)

if k 太大.

$k \to 1/\Delta t$.

Verlet integration

- compute position in next frame from position in current & previous frame.

$p(t+\Delta t) \cong p(t) + (v_t + a(t)\Delta t)\Delta t$.

$= 2p(t) - p(t-\Delta t) + a(t)\Delta t^2$.

with viscous damping

$p(t+\Delta t) = (2-d)p(t) + (1-d)p(t-\Delta t) + a(t)\Delta t^2$.

Easy to implement, stable error: $O(\Delta t^4)$

Constraint Satisfaction.

Verlet integration 时.

- find the closest position that satisfies the constraint → move it there.

distance constraint.

- view distance constraint as a spring → solve for equilibrium position → move ~~both~~ both.

  (→ = ← position).

- compute different b/w desired len (o). → move along o. → weight motion with mass.

T Ragdoll physics in Hitman.

→ model as particles (at joints) + massless rods (bones).

→ verlet integration.

特点 good dead body. 缺点 just same animation

~~USEMO~~

Extended bodies

Using more particles. → connecting them with springs to hold them together

~~[illegible]~~

Use Verlet as the integrator.

→ for soft body physics & clothes simulation.

Soft body physics.

T for RB. 如果 particles 够 RB. ∴ spring suffer 太多 / expensive 计算量. ⊥

RB dynamics.

add rotation degrees of freedom to object's state.

track → linear pos, mass, momentum, angular pos, mass, momentum.

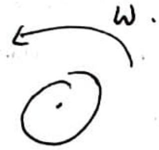
Center of mass motion.

position is $\frac{1}{M}\int \rho(r)\,dr$. moves based on net momentum of all mass.

Rotation.

{ rotation
{ center of mass.

change angular velocity.



Angular momentum 角动量 — density of mass.

$$L = \int r^2 \rho(r)\,\omega\,dr = I\omega$$

$$I = \int r^2 \rho(r)\,dr$$

角动量惯性

Torque. : rotational equivalent of force. 扭矩

$$\tau = \frac{dL}{dt} = \frac{d(Iw)}{dt} = I\left(\frac{dw}{dt}\right) = I\alpha$$

Force to rigid body.

where & what direction.

- change of linear & angular momentum.
  - apply center. (linear)
  - torque about (center). (to angular).
  - collision at contact point. (changes both. based on contact point & momentum of colliding body).

Articulated Bodies.

objects aren't rigid in real life, they're springy.

Penalty methods

too stiff 僵硬. explode. not stable.

not stiff. weakly satisfied.

Ex. crates 箱子 stacking.

spring btw that push them apart.

不会 fall throw. 会掉). spring大.

State space.

each particles or body has a state.

- position.
- velocity.
- orientation.

one cannot pass through 2 anywhere is fine except occupy the same pos.

Constrain resolution

integrating the state.

need to resolve constrain & check valid state.

projection method.

want here.

- check state
- valid?
- find nearest x
- move ~~there~~ to x.

8. Collision Detection.

about volume not surface.
↑ collision geometry.   ↑ rendering geometry.

Seperating axis theorm.

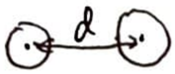
convex & concave.  convex / concave.

If two shape are convex, there has to be a seperating line.

Primitive shapes. – for collision.

have in advance: symmetrical shape.
(circles, spheres, circle/sphere-swept volumes (SSV), boxes).

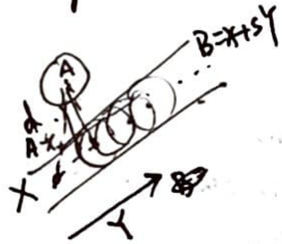
Sphere intersection test.



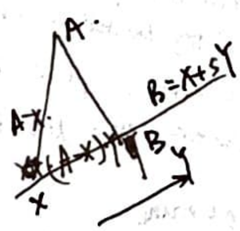
- get d.
- compare sum of radii 半径 $||A-B|| < A_r+B_r$.
- compare squared distances. $||A-B||^2 < (A_r+B_r)^2$.

要画个中间 seperatin line for [illegible]
seperating axis 为 这个的 垂直线.

Sphere & cylinder.



Swipe the sphere along the line.

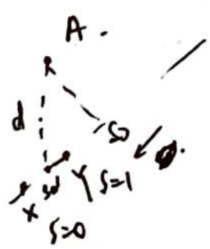


$((A-X)\cdot Y)\cdot Y$ = offset of bring X to point B.

$\Rightarrow ||A-((A-X)\cdot Y)Y-X||$

point B = $X + (A-X)\cdot Y$

点B: $s = \dfrac{(A-X)\cdot(A-Y)}{||X-Y||^2}$ min d = $\dfrac{||(A-X)\wedge(A-Y)||}{||X-Y||}$ $B = X+s(Y-X)$.



$s_0 = \dfrac{(A-X)\cdot(A-Y)}{||X-Y||^2}$.

real.
$s_0' = \min(1, \max(0, s)) = \min\left(1, \max\left(0, \dfrac{(A-X)\cdot(A-Y)}{||X-Y||^2}\right)\right)$.

$d = ||A-(X+s_0'Y)||$ $||A-(X+s_0'Y)|| < A_r+B_r$.
or square both sides.

Bounding Box.

Axis aligned AABB

< Oriented OBB

just integrals.

AABB Intervals A, B disjoint if

$A_{min} > B_{max}$ or $B_{min} > A_{max}$

(amin.x > bmax.y || am ----).

Separating axis theorem.

if two convex non-intersecting.

must have some axis along which they're disjoint.

Special case

shapes are polygons.

Contact point determination:

for 물리 engine → physics.

* if large moving slowly, 매 update distance info

is small & finite. 충돌 예측.

↳ dynamic collision detection.

eg. bullets. swept-y cylindrical volume

detection if volume collide.

Broad-phase collision

intersection 자체 expensive.

every pair of objects · $O(n^2)$.

& may goes big.

want to:

rule out most pairs.

not tests for intersection

· broad-phase: prune the space of pairs to test

narrow-phase: detailed.

- Collision layers.

which allow to collide with one another

Sweep & prune.

- Sort all object $O(n \log n)$.
- sweep an imagery plane from one end to the other. $O(n)$.
- test it against all obj intersecting the plane. $O(n)$.
- still $O(n^2)$ worst case but better in avg case.

↑ all obj in same place

Space tree:

Bsp. kd.

⋮

$O(n \log n)$

12 Model 3D. world

In 2D.

$u \wedge v = \left| \begin{bmatrix} u_1 & u_2 \\ v_1 & v_2 \end{bmatrix} \right|$.

$= u_1 v_2 - u_2 v_1$.

3D.

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \wedge \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{bmatrix}.$$

~~Projection~~.

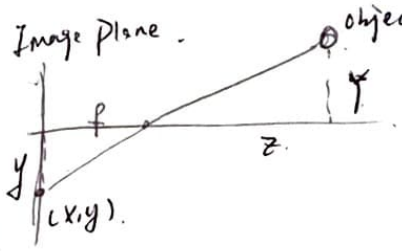
Projection.

- Orthographic projection
  - 不管 z. 只看 x.y.
  - 和 scalar work to help fit scene onto the screen.
  - 为了看得(远处中)/大视角.
  - Scalar $\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
  - this is linear transform.
- perspective projection
  - real cameras.
  - divide z.
  - 和 scalar factor (focal length).
  - ~~$y/f = Y/z$~~ ~~$y = fY/z$~~.

    $(x.y) = (fX/z, fY/z)$.
  - perspective projection

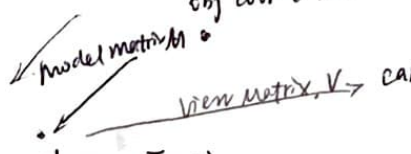
    $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ move z to w
  - perspective projection

    $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{bmatrix}$



Y - height of obj
Z - depth.
y - "height" of projection.
f - focal length.

Chained coor System.



rendering:

- 3D model in object local coor.
  with model coor changed by Model matrix M.
- but we render from camera's viewpoint
  from world to ~~obj~~ camera by view Matrix V.

$\Rightarrow VM\beta. = \lambda_2$ render coor

- project matrix P. (camera → screen coor)
  定义 camera scale, 视角...

(PVM)p.
└ child depend on parent 的 matrix multiply 完以后算出 matrix)

[3] Model 3D shape.

representation of the shape < curves in 2D
surface in 3D.

Approximating a curve
$f: \mathbb{R}^n \to \mathbb{R}$. & any input x_0 for f. then's Df_{x_0}
: Let $g(x+\Delta x) = f(x_0) + Df_{x_0}(\Delta x)$.
$|f(x_0+\Delta x) - g(x_0+\Delta x)| = o(\Delta x^2)$.
$f(x_0+\Delta x) \cong f(x_0) + f'(x_0)\Delta x + \frac{1}{n!} f^n(x_0)\Delta x^n$.
第 n^{th} polynomial error $= o(\Delta x^{n+1})$.
区域足够小时

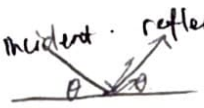
Approximate a surface
polygonal meshes - vertices, edges, faces.
triangle mesh: easy for GPU to handle this.
~~Triangle mesh~~
modeling technique.
- 3D scanning.

Rendering Related to Material.

- transmission of light.
- from a light source, to a camera.
- interact with surface along the way.

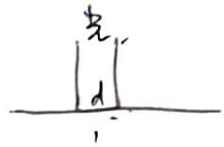
— ~~Surface~~ reflection.

specular



— Lambertian reflection.

~~Surface reflection~~



光花不同大
照亮面积不同

Surface normals.

$\frac{d}{\cos\theta} = \frac{d}{L \cdot n}$.

normal:

$L_n = n(L \cdot n)$

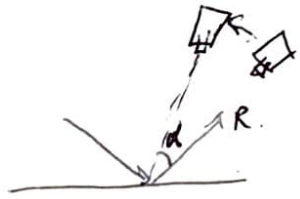
tangential:

$L_T = L - L_n$.

$\Rightarrow L = L_n + L_T$.

$R = L_T - L_n$.

$= L - 2(n(L \cdot n))$.



dot camera · dot reflected = $\cos\alpha$

- bright if near reflect ray
- get darker slowly

→ make $\cos\alpha \to (\cos\alpha)^2$.

Norm in 3D.

are points have normals.

flat shading → all points on a face have same normal. boundary noticeable.

normal interpolate → interpolate normal within a triangle.

→ each pixel in a face recalculate its normal.

algo: — use average of normals of each vertex.

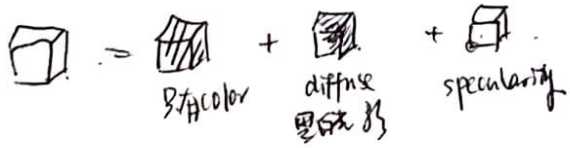
— weight distance from the pixel.

— closer pixel get more influence.

phong shading — algo: ~~Gouraud~~

— specular & diffuse component with normal interpolation.

— ambient component, constant added to all pixels, acts like a ~~torch~~ full light.



Solve:

- Geometry.
  - vertices of the mesh.
  - triangles formed by vertices
  - tagged with . normals
    - color info
    - texture coord.
    - exotic infos. (weights ...) ↓ skin animation.
- material (shader program).
  - diffuse color
  - specular color
  - specular hardness
  - ambient color.
  - texture map.
  - ~~bump map~~ exotic info - bump map / specularity map.

texture mapping

instead of dye each polygon, it may take triples to make realistic object

Diffuse texture map.

diffuse color read from texture map for each pixel

specularity map

~~有些物体~~ → ~~部分~~ seperate specularity map. to show some part selectively shiny

normal mapping.

make low polygon model → high polygon model.

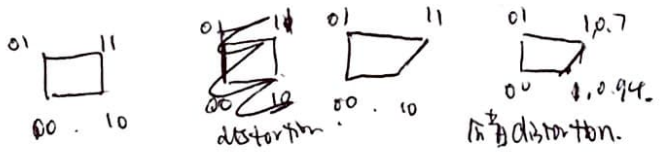
artist should paint changing normals over single polygon.

to show lighting → high polygon feel.

packed texture map

→ save memory & time.

texture map polygon rendering



The curse of topology. eg. 人四全平面.

stretch some parts.

squish others.

cut ~~note~~ holes (有时间)

Mesh → 线点 vertices. in a lot of direction.

texture map → for character (packed texture map).

mapped mesh. ~~texture map~~ 网

texture map 和 mesh map在一起.

画纹 distortion 等处理, 对于 camera 位置.

多: transparency.

「opaque 只看最上一层.」

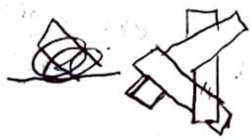
透明: - have to render all points.

- combine effects.

还是从最下到最上

3.: depth first algorithm

in film industry done by sorting individual polygons by depth.



- have to situations where a polys overlap in depth.
- have to split polygons.

but not practical in game.

15. Rotation in 3D.

[illegible]帧 transform: 定 Key frame 在人 平面上. translate. rotate ~~stat~~ skretch, squish. bend.

- different obj have different configuration spaces.
- have 3 Dof. can only translate it.

回 rigid bodies have much simpler configuration space

- translate & rotate.

变 - position

- direction.

pose = translation + rotation.

rotation → direction (axis). magnitude (angle).

rotation.

① Rodrigues'

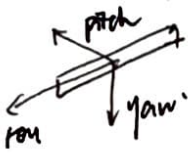
$$V_{rot} = v\cos\theta + \underbrace{(w\times v)}_{\text{towards new rotation}}\sin\theta + w\underbrace{(w\cdot v)}_{\text{re-add the height}}(1-\cos\theta)$$

Scale down "

$$\frac{a\times b}{|a\times b|} = \frac{a\times b}{|a||b|\sin\theta}.$$

优：simple. compact. efficient　　缺：hard to compose rotation. not

cup down → yaw.
left right → pitch.
front/back → roll.



优：easy understand. compact. easy to compute.　　缺：determine YPR angle 不易. 两个axis会合. hard to compose. can't interpolate. gimbal lock.

Euler angles.
优：compact　angles aren't order-dependent
缺：not composable. not interpolatable. not intuitive. gimbal lock.

For rotation matrix.

$$V_{rot} = V\cos\theta + (w \times V)\sin\theta + w(w\cdot V)(1-\cos\theta)$$

$$w \times V = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix} V = [w]_\times V.$$ Cross-product matrix.

$$V_{rot} = V\cos\theta + (w \times V)\sin\theta + w(w\cdot V)(1-\cos\theta).$$

优：no gimbal lock. easy to compose rotation　缺：more memory. can't interpolate.

Euler-Rodrigues Form (Isomorphic Quaternion)

$$R_{w,\theta} = I\cos\theta + [w]_\times \sin\theta + (1-\cos\theta)[w]_\times^2$$

$$= \begin{bmatrix} \cos\theta + w_x^2(1-\cos\theta) & w_x w_y(1-\cos\theta) - w_z\sin\theta & w_x w_z(1-\cos\theta) + w_y\sin\theta \\ w_y w_x(1-\cos\theta) + w_z\sin\theta & \cos\theta + w_y^2(1-\cos\theta) & w_y w_z(1-\cos\theta) - w_x\sin\theta \\ w_z w_x(1-\cos\theta) - w_y\sin\theta & w_z w_y(1-\cos\theta) + w_x\sin\theta & \cos\theta + w_z^2(1-\cos\theta) \end{bmatrix}$$

$$= \begin{bmatrix} a^2+b^2-c^2-d^2 & 2(bc-ad) & 2(bc+ac) \\ 2(bc+ad) & a^2+c^2-b^2-d^2 & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & a^2+d^2-b^2-c^2 \end{bmatrix}$$

$a = \cos\frac{\theta}{2}$
$b = w_x \sin\frac{\theta}{2}$
$c = w_y \sin\frac{\theta}{2}$
$d = w_z \sin\frac{\theta}{2}$
$a^2+b^2+c^2+d^2 = 1$

Summary:

	优	缺
rotation vector:	simple; compact; efficient to compute	compose; interpolate.
YPR	understand; compact; easy compute	angle determination; communitive; compose; interpolate gimbal lock
Euler.	compact, aren't order dependent.	compose, interpolate, intuitive, Gimbal lock.
Rotation Matrices	compose, compose with other, derive from any.	memory, interpolate.
Quaternions.		
slerp.		

~~Euler-Rodriguez Form. (Just an isomorphic Quaternions~~

~~$M_{w,\theta} = I\cos\theta + [w]_x \sin\theta$~~

Complex number.

16. Asychronous.

detecting completion.

polling : simple, complex.inefficient 如果总在做不好了.

interrupts/callback : efficient (不同thread). messy (what thread, 怎么cancel. exception怎么办, [illegible])

message passing : 有message ~~queue~~ queue. (主流的方法)

有message loop dequeue from message queue.

Parallel.

Aschronous calls ~~race condition~~ (HeisenBug)

L 有时有有时没有 → time dependent.

mutual ~~exp~~ exclusion ~~[illegible]~~ 一次有一个thread用某个variable.

会堵 & ~~expensive communication~~ expensive.

thread x
wait n until available
⋮ ⋮ operations.
n available again.

Cooperative multitasking : one thread, shared btw tasks.

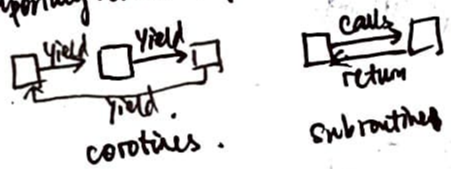
volunteerly yield CPU to let other task run.

efficient, no race condition, starve other tasks, 要用 aschronous call 结果slow.

还是只用了 one core.

Coroutines.

Subroutines can temporally return. & pick up later. → yielding.



Implementation.

class : PseudoCoroutine.

method : override continue.

Task.

Add into list of tasks.

Continue.

ticking.

Finite State Machine (FSM)

State —.

update() {

switch state }.

Starvation: 有太多loop. 别的stop了都.

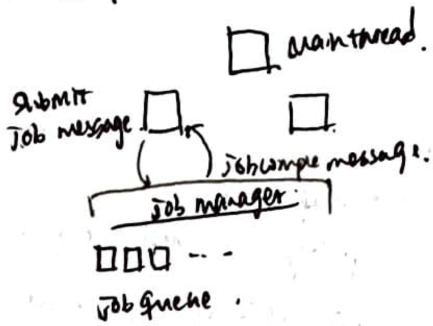
(Realtime Constraint).

→ 把FSM多个节点.

多个if statement check会很多到 不一样等.

Job System.

when have a lengthy computation, one thread for slow computations.



Coroutine in Unity.

IEnumerator.

yield return.

Rendering 17

不画: 一条线上, offscreen, normal 和 camera away from.

画: 每个scanline → 每个fragments.

compute depth

for each sample.

compute lighting ...

有的不画的: culling. 看不见不画.

parallelism. 分开的工作 多个 processor.

simplification. 没有必要画的就省.

→ fragment level. offscreen不画.

normal 反的不画.

Z 在后的不画 → Z-buffering.

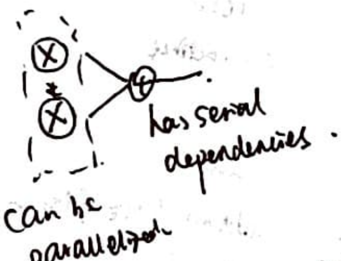
object level. frustum culling (frustum 之外的不画)

occlusion culling. 在后面的

Scene graph culling.

paralism.

Parallelism.



MIMD (multi instruction, multi data)

SIMD (single instruction multi data)

Pipelining 如果都有2部 (one processor on 1st; the other on next)

Renderer Structure (pipelining).

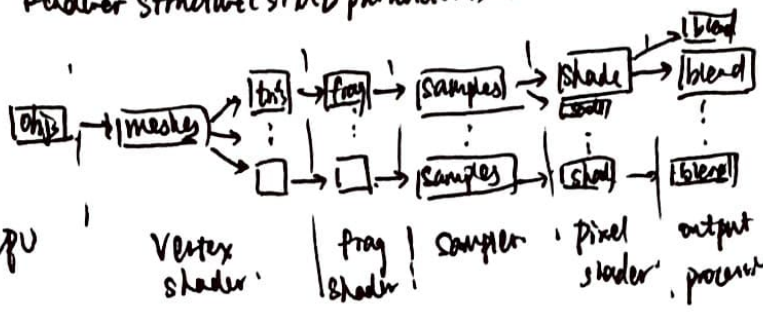
obj → meshes → tris | → frags → samples → shade → blend

queue.

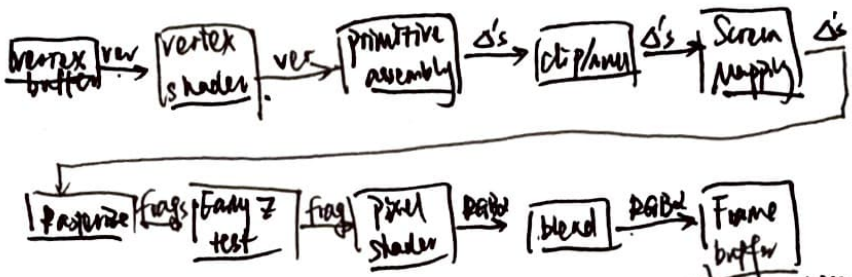
CPU.

video card.

Renderer Structure (SIMD parallelism)



GPU pipining.



vertex buffer: Stored once in GPU memory by CPU repeatedly, specifies triangle vertices of meshes to be drawn.

vertex shader: Stored once in GPU by CPU repeatedly, specifies transformation of vertices & vertex attrs.

pixel shader: Stored once in GPU by CPU, specifies computation of color based on vertex attrs.

shader parameters: stored in GPU by CPU, possibly updated frame to frame or object to object, specifies inputs to shader that don't vary from Δ to Δ

render states: Set by CPU, varied during frame, but as little as possible, specify behavior of configurable, but not programmable pipeline stages.

render target: Set by CPU, specifies which region of GPU memory to use for frame output.

GPU performance issue.

pipeline stalls. 导致暂停. vertex processor 等 if no pixel processor available.
pixel processor 等 if vertex processor 还没准备好 vertex.
change 导致 if 为 state 改变.

Overdraw.

memory contention

locality. so using the same vertex twice a row is fast.

CPU performance issues.

Batching: 将 assemble into & send command packet → batch
building command packet batch is expensive.

bandwidth issue.