PICKING IN 3D

SUNDARAM RAMASWAMY

TYPICAL Q&A SCENE

```
char foo = 1;
std::cout << +foo;  // removing '+' prints a funny character: ☻
```

Asker: Shouldn't **+** be a no-op?

… wall of speculations and arguments by *Joe Coders* …

Language lawyer: According to *C++14*, §5.3.1/7:

> *[…] the unary **+** operator […] **integer promotion is performed** […] result is […] promoted operand.*

Asker: Of course! Dunno how I missed it, thanks ¯_(ツ)_/¯

*Upvotes all around; **no** arguments.*

(NIT) PICKING DEFINITIONS

Take your pick.

- Selecting carefully from a group
- Harassing with constant criticism

Selecting something, in 3D, using a pointing device

WHY POINTING DEVICES? WHY 3D?

- **Keyboard selection**: nothing special, algorithmically
  - Iterates over a pre-determined set of objects
  - Usually not called *picking*; just *selecting*
  - Other imprecise devices are similar e.g. joystick
- **2D picking**: just a bunch of point-in-polygon tests
  - Entities can overlap? Tackle Z-order as phase two
  - Subset of 3D picking in some sense

MOTIVATION

- “Hey, consoles have joysticks, not pointing devices!”
  - Sure, but game design tools use mouse extensively
- Picking happens on every click
  - Better to know the underlying math
- **Curiosity**: challenge of selecting in 3D with a 2D input
  - Intuitive in 2D but unnatural in 3D
  - How is the loss in dimension compensated?
- Fun to learn!

OVERVIEW

- **Rendering**: mapping 3D data on to a 2D canvas
- **Picking**: mapping a 2D point to a 3D world
- Inverse operations in some sense
- Need a way to run the pipeline backwards

$$View \xrightarrow{projection} Clip \xrightarrow[z]{divide} NDC \xrightarrow{viewport} Screen$$

PROBLEMS

Pipeline not always reversible — mathematical issues faced:

- **Singularity**
  - Projection matrices are usually rank deficient; non-invertible
- **Non-linearity**
  - Perspective division makes inversion process non-linear
- **Aliasing**
  - Multiple points in 3D are represented by one point in 2D
  - Which one to transform with the inverse matrix?

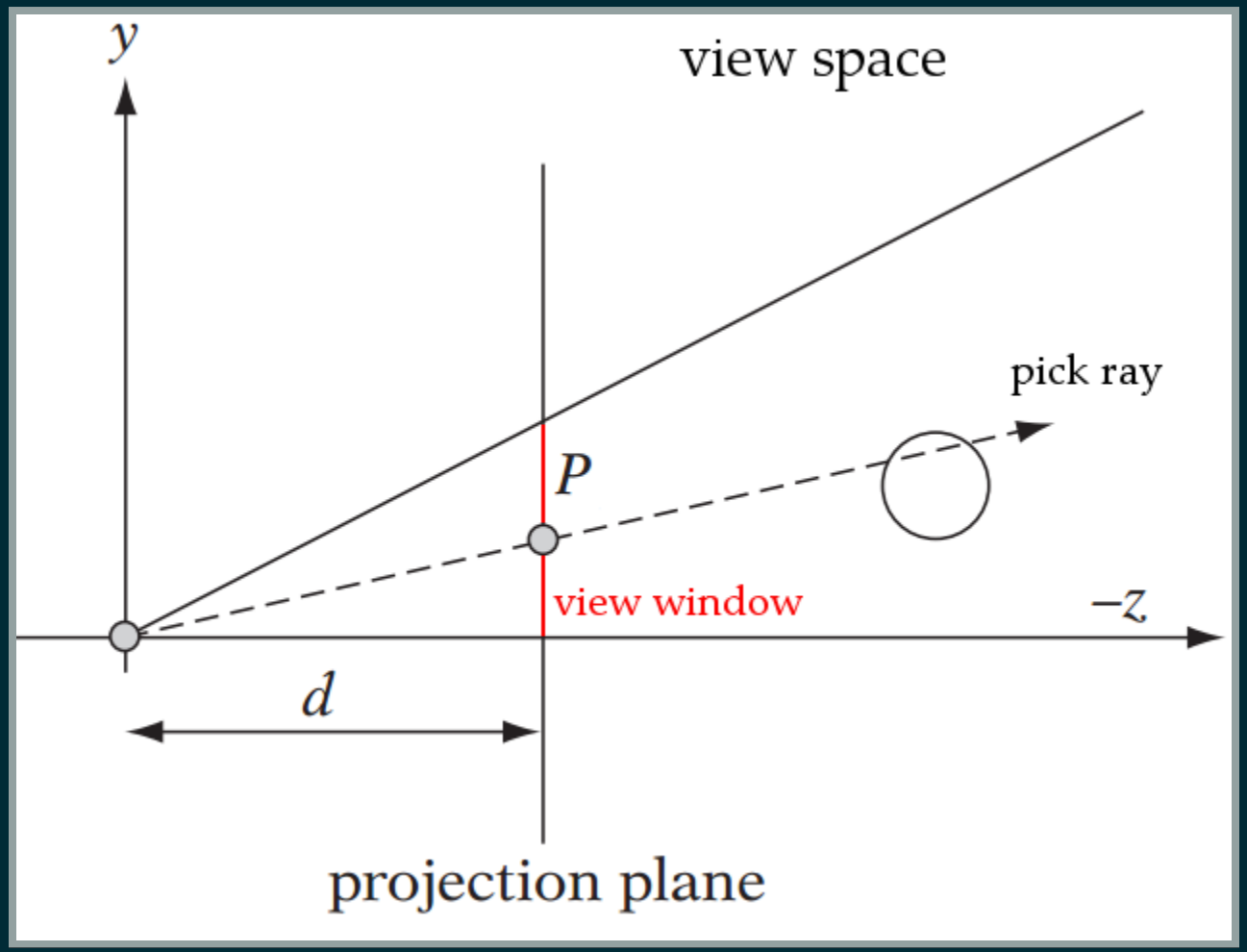
ALIASING

- Need Screen (2D) $\mapsto$ **Normalized Device Coordinates** (3D)
- NDC is *standardese* for **Canonical View Volume**
  - Visible volume that gets rasterized finally
  - NDC $(x, y, z) \in [-1, 1]$
- **Ray casting** remedies aliasing!
  - Shoot a (pick) ray from view origin along all aliased points
  - Closest intersecting object is picked
  - Challenge is to find ray direction
    - Specifically another point on the ray

NON-LINEARITY

- Perspective division cannot easily be undone
- Find one aliased point (P) to calculate pick ray direction
- One such point would be on **view window** in view space
  - Rectangle on projection plane mapped to viewport
- $\therefore P_z = d$, focal length: `distance(origin, view plane)`
- Circumvents undoing perspective division neatly
- P_x and P_y are still unknown

PICK RAY

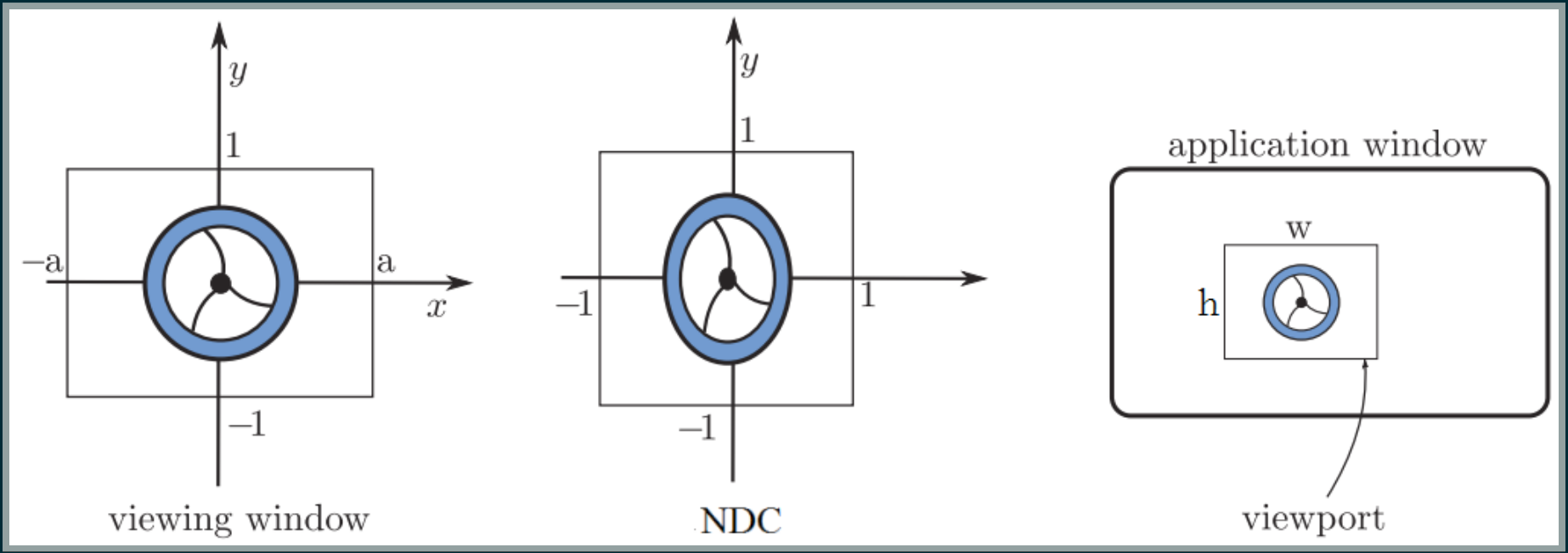


Credit: James M. Van Verth, Lars M. Bishop
Essential Mathematics for Games and Interactive Applications

SCREEN ⟶ VIEW

- Pick ray constructed in view space but got x_{scr} and y_{scr}
- Find maps to transform Screen → NDC → Clip → View
- Clip space can be ignored
  - 4D homogeneous space to set up z in w for perspective divide
  - Used only for clipping otherwise
- Essentially find **Screen → NDC → View** (with z dropped)
- Rectangle mapping: $[w, h] \mapsto [2, 2] \mapsto [2a, 2]$
- Should require only scaling and translation

SPACES



Credit: Ganovelli, Corsini, Pattanaik, Benedetto
Introduction to Computer Graphics ~ A Practical Learning Approach

SCREEN $\longrightarrow$ NDC

- Rendering mapped NDC cube to screen space cuboid such that
  - $x_{scr} \in [-w/2, w/2]$
  - $y_{scr} \in [-h/2, h/2]$
- Rectangle dimensions do not vary with depth for both volumes
- Straight forward inversion; one rectangle to another

$$\boxed{x_{ndc} = \frac{2x_{scr}}{w} - 1} \qquad \boxed{y_{ndc} = -\frac{2y_{scr}}{h} + 1} \qquad (1)$$

NDC $\longrightarrow$ VIEW

- Rendering mapped view frustum to NDC cube
  - Rectangle dimension varies based on depth in frusutm
  - Already chosen z = focal length; view plane
- View window dimensions, where a is aspect ratio

$$x_v \in [-a, a] \qquad y_v \in [-1, 1]$$

- Map one rectangle to another like before

$$x_v = a x_{ndc} \qquad y_v = y_{ndc} \qquad (2)$$

- Final **Screen** $\mapsto$ **View** by substituting (1) in (2)

$$\boxed{x_v = \frac{2a x_{scr}}{w} - a} \qquad \boxed{y_v = -\frac{2y_{scr}}{h} + 1}$$

HIT TESTING

- Hit test with pick ray in view space
  - Apply inverse camera transform if testing in world
  - $M_{w \to v}^{-1}$ usually cheap; rigid body transform inverse
- Shoot ray from `near` plane
  - Avoids hitting objects before near plane
- Sort by t; pick first object
- Rides on spatial data structure facility provided by engine

ALTERNATIVE APPROACH

Color-based picking

- Flat shade all selectables with unique color on framebuffer
- Read color from buffer at click point
- Get object from color-to-object map
- Good
  - Easy to implement; almost no math needed :)
  - No dependency on engine or complex data structures
- Bad
  - Coarse; cannot find exact triangle clicked
  - Needs unique color generation for objects coming into view
  - May be slow: needs flushing to get rendered framebuffer

Q & A

THANK YOU!