Common interview topics for programming interviews.

by Warwick New

Introduction

When applying for AAA programming positions, especially those of a more technical nature, there are going to be some some barrier to entry questions that can be the deciding factor between you and another applicant.

Task

In this session I want you guys to see how relevant each of these skills are to what you want to do in industry and based on that make decisions on getting an understanding of how these aspects work under the hood.

Optimisation

From personal experience, if you are going into anything more technical than gameplay programming, you are going to find that interviewers are going to want you to explain how you’ve optimised code in the past.

Topics From my experiences

Personally I’ve had technical tests that expected me to make use of **concurrency, abstraction & chunking, pointers** and knowledge on **CPU memory caching** to reduce simulation times from seconds per frame to milliseconds per frame.

Topics From Jakes experiences

Jake Meaker has told me about his application to a AAA studio that required him to explain low level C concepts such as **bit shifts, binary operations, and hex code operations**.

Optimisation

Questions to ask yourself

- Can I use concurrency safely in my programming language of choice?
    - *Possible task*: use concurrency to optimise a simple project like drawing the Mandelbrot set.
- Can I use objects to abstract data not just for readability purposes but also for performance purposes.
    - *Possible task*: Use level chunking to optimise a simulation like boids or one of your games with a large map.
- Do I know how memory is allocated and deallocated in ram.
    - *Possible task*: Tinker with bit shifting and small optimisations based on making use of CPU features like SIMD.
- *If advertising C++ experience*, can I explain how and why I'd choose to use a pointer in a program.
    - *Possible task*: Create a program that makes use of pointers in a useful manner.

Maths - Linear Algebra

Cross Product, Dot Product.

These are two extremely common functions that are extremely useful for common 3D operations you may have to perform in game engines.

> *The dot product is a simple yet extremely useful mathematical tool. It encodes the relationship between two vectors' magnitudes and directions into a single value.*
>
> — *Allen Chou*
> *Naughty Dog*

Maths - Linear Algebra

Cross Product, Dot Product.

Dot Product

- We can tell if the angle difference is obtuse acute perpendicular or facing the same way really easily.
- We can tell the magnitude difference between two vectors as well
- You can use dot products to work out if a position is in a filed of view or what direction something is entering a trigger(Only trigger event when player leaves room not enters etc).

Cross Product

- Used to generate vectors that are perpendicular to two vectors.
- Useful for reflections of beams of light ricochet getting the sideways vector of a object based on it’s forward vector. (what way should A&D move with WASD)

Maths - Cross Product, Dot Product. Questions to ask Oneself

- Can I describe what a Dot Product/Cross Product is?
  - *Possible task*: Write yourself a refresher document before you might interview.
- Can I point to examples of having used these algorithms.
  - *Possible task*: If you haven't used these for anything simulate a beam of light puzzle with reflections.
- Can I explain how I'd use one in a given context.
  - *Possible task*: Try to use them in multiple contexts until you understand their usefulness.

Maths - Linear Algebra

Matrices

You may not need to know too much about these but knowing the basics can be very handy if you're performing a lot of translations. Mostly useful for graphics and physics programmers.

- Matrices are represented like 2D arrays of numbers.
- A single matrix can represent a change in a vector using: translation, rotation, and scaling in one piece of data.
- Fun fact matrices are used to translate objects in world space to a flat plane when game engines render scenes.

Physics

Most of you guys have interacted with a lot of physics systems in game engines already so you should have an intuitive understand of most of this stuff. But can you put it into words.

- Can you discuss how velocity, rates of change, and gravity is simulated. (Newtonian Mechanics)
    - *Possible task*: Work through a bouncing ball simulation in a simple 2d sandbox and annotate how your code relates to newtons laws.
- Can you discuss how to make objects bounce off angled surfaces (Cross/Dot product could useful here)
    - *Possible task*: Create a spline for predicting where a cannonball will be fired and bounce in a simple 2D environment.
- Collision Detection
    - *Possible task*: Research common collision detection methods and implement some simple ones.

Maths/Physics - Euler Rotations vs Quaternions

Most of you have tinkered with rotating objects in game engines, Rotating using Euler’s roll, yaw, and pitch. Looking at quaternions. Just remember that these things can happen.

- Eulers rotations can run into Gimbal lock, (An object rotation get’s stuck on one axis)
- Quaternion rotations avoid this issue.

Networking Stack

If you're doing anything with networks or distributed systems it helps to know these things.

This is the web stack

image::images/http-layers.png[https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview]

Networking Stack

Points to remember.

- Most video game netcode is built on top of UDP and TCP which communicate through sockets. (Though some projects use web sockets nowadays)
- Sockets are connected to the web through ports (Minecraft servers use port 25565 by default)
- The sever will likely need to use concurrency to handle multiple real time connections on these sockets.
- RPC stands for remote procedure call and tells the server to run a function on it’s side, often used for triggering events in multiplayer games.

Addendum - Bouncing raindrops with physics (Website only)

```
// create background element
let canvas = document.createElement("canvas");
document.body.appendChild(canvas);
canvas.style.cssText = `
background: var(--background);
position:fixed;
left:0;
top:0;
z-index:-99999;
`
// set it's size
canvas.width = window.innerWidth;
canvas.height = window.innerHeight;

let context = canvas.getContext('2d');

// this will be populated and used later
drawableObjectList = [];

class Splash {
  constructor(distance, x, y) {
    this.distance = distance;
    this.x = x;
    this.y = y;
    this.time = 0;
  }
  draw() {
    context.fillStyle = "#2a232300";
    context.strokeStyle = `rgba(42, 35, 35, ${this.distance + 0.1})`;

    let splashSize = this.distance * (this.time / 100);

    context.beginPath();
    context.ellipse(this.x, this.y, 100 * splashSize, 50 * splashSize, 0, 0,
Math.PI * 2);
    context.fill();
    context.stroke();

    if (this.time > 100) {
      drawableObjectList = drawableObjectList.filter((item) => { return item
!== this });
    }

    this.time += 1;
  }
}
// this is the object that'll look cool
class Drop {
  constructor() {
    this.x = Math.random() * window.innerWidth;
    this.y = Math.random() * window.innerHeight;

    this.distance = Math.random();
    this.velocity = [(Math.random()-.5) * 5, this.distance + 0.1];
  }
  draw() {
    context.fillStyle = `rgba(255, 255, 255, ${this.distance + 0.1})`;
    context.strokeStyle = `rgba(255, 255, 255, ${this.distance + 0.1})`;

    // draw point
    context.beginPath();
    context.moveTo(this.x, this.y);
    context.lineTo(this.x, this.y - this.distance * 100);
    context.stroke();
    // simulate gravity
    const gravity = 1.05;
    // move point
    this.y += this.velocity[1] ;
    this.x += this.velocity[0] ;
    // Apply gravity to our velccity.
    this.velocity[1] += gravity;

    // keep point in bounds
    if (this.y > window.innerHeight - (1 - this.distance) * 300) {
      // Add splash
      drawableObjectList.push(new Splash(this.distance, this.x, this.y))

      // Apply bounce based on floor
      this.velocity[1] = this.velocity[1] * -1 * 0.95
    }

    // keep point in bounds
  if (this.x > window.innerWidth || this.x < 0) {
      // Add splash

      // Apply bounce based on wall
      this.velocity[0] = this.velocity[0] * -1
    }
  }
}

// populate element array
for (let i = 0; i < 100; i++) {
  drawableObjectList.push(new Drop());
}

function loop() {
  // Loop and clear frame
  requestAnimationFrame(loop);
  context.clearRect(0, 0, window.innerWidth, window.innerHeight);

  // Draw objects.
  drawableObjectList.forEach((point) => {
    point.draw();
  });

  // Handle page size change
  if (canvas.width != window.innerWidth) {
    canvas.width = window.innerWidth;
  }
  if (canvas.height != window.innerHeight) {
    canvas.height = window.innerHeight;
  }
```

```
}
loop();
```