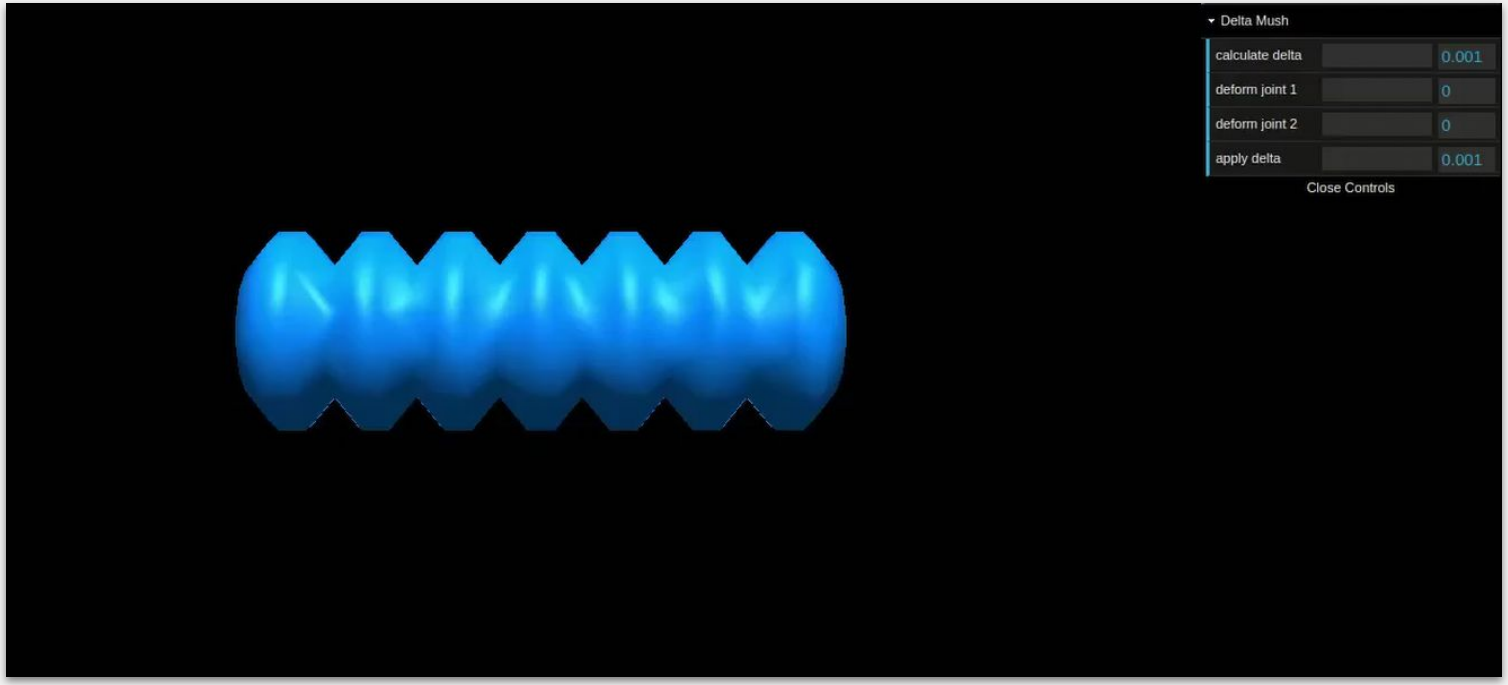
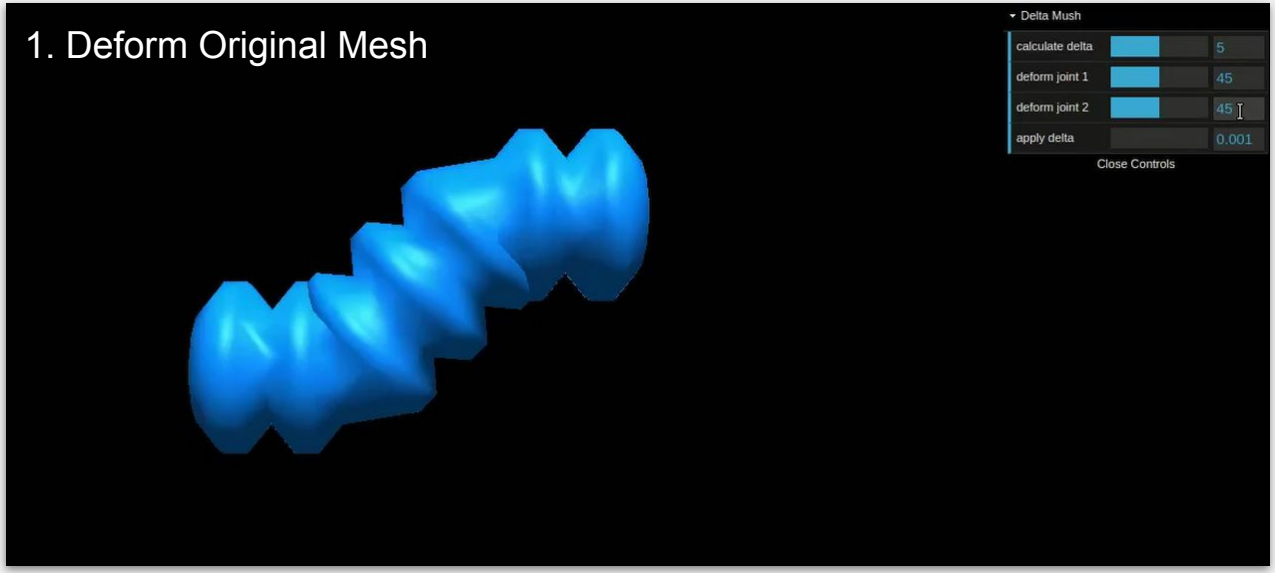
Delta Mush - Deforming Mesh Structures with Three.js



Implemented paper https://dl.acm.org/doi/10.1145/2633374.2633376

Delta Mush - Deforming Mesh Structures with Three.js

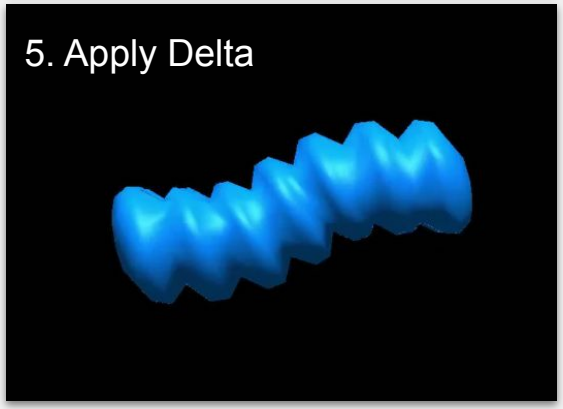
1. Deform Original Mesh



2. Apply Laplacian Smoothing
3. Calculate Rest Coordinate System + Vertex Offset

4. Smooth Deformed Mesh

5. Apply Delta



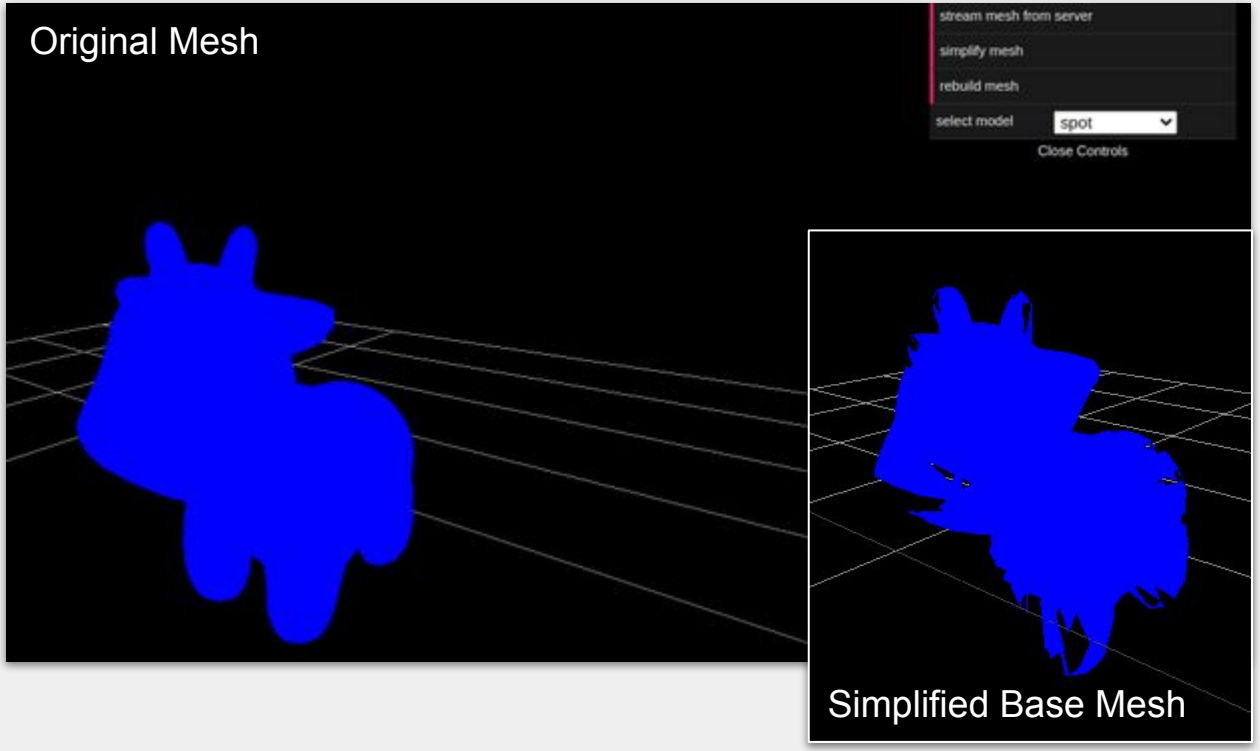
Delta Mush - Deforming Mesh Structures with Three.js

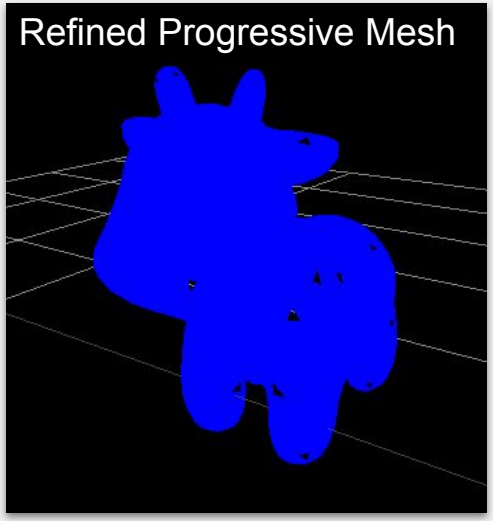
```
738   delta(weightType: WeightType, timeStep: number, smoothStep: number) {
739     this.smoothDef(weightType, timeStep, smoothStep);
740     let vMatrix = DenseMatrix.zeros(4,1);
741
742     this.verts.forEach(v => {
743       let current = this.currentCS(v.idx);
744
745       vMatrix.set(this.vertsOffset[v.idx].x, 0, 0);
746       vMatrix.set(this.vertsOffset[v.idx].y, 1, 0);
747       vMatrix.set(this.vertsOffset[v.idx].z, 2, 0);
748
749       let d = current.timesDense(vMatrix);
750
751       this.vertsfinal[v.idx] = new Vector(0,0,0,0);
752       this.vertsfinal[v.idx].x = d.get(0,0);
753       this.vertsfinal[v.idx].y = d.get(1,0);
754       this.vertsfinal[v.idx].z = d.get(2,0);
755     });
756
757     //set final vertex positions with delta
758     this.verts.forEach(v => {
759       v.position.x = (this.vertsfinal[v.idx].x);
760       v.position.y = (this.vertsfinal[v.idx].y);
761       v.position.z = (this.vertsfinal[v.idx].z);
762     });
763   }
```

Delta Mush - Deforming Mesh Structures with Three.js

```
701  smoothDef(weightType: WeightType, timeStep: number, smoothStep: number) {
702    //Build f(t)
703    let f = DenseMatrix.zeros(this.verts.length, 3);
704    //fill matrix
705    this.vertsDefOrig.forEach(vert => {
706      f.set(vert.position.x, vert.idx, 0);
707      f.set(vert.position.y, vert.idx, 1);
708      f.set(vert.position.z, vert.idx, 2);
709    });
710
711    //build the mass matrix
712    let T = new Triplet(this.verts.length, this.verts.length);
713
714    this.verts.forEach(v => {
715      T.addEntry(1, v.idx, v.idx);
716    });
717    this.verts.forEach(v => {
718      //multiplying with 100 reduces issues with small cells
719      T.addEntry(v.voronoiCell() * 100, v.idx, v.idx);
720    });
721
722    let M = SparseMatrix.fromTriplet(T);
723    //build Laplace weight matrix for the given weightType
724    let W = this.laplaceWeightMatrix(weightType);
725
726    //solve linear system (M - tλW)f' = Mf using a Cholesky solver
727    let F = M.plus(W.timesReal(1000*smoothStep));
728    let fQM = F.chol().solvePositiveDefinite(M.timesDense(f));
729
730    //Update the position of mesh vertices based on the solution f'
731    this.verts.forEach(v => {
732      v.position.x = fQM.get(v.idx, 0);
733      v.position.y = fQM.get(v.idx, 1);
734      v.position.z = fQM.get(v.idx, 2);
735    });
736  }
```

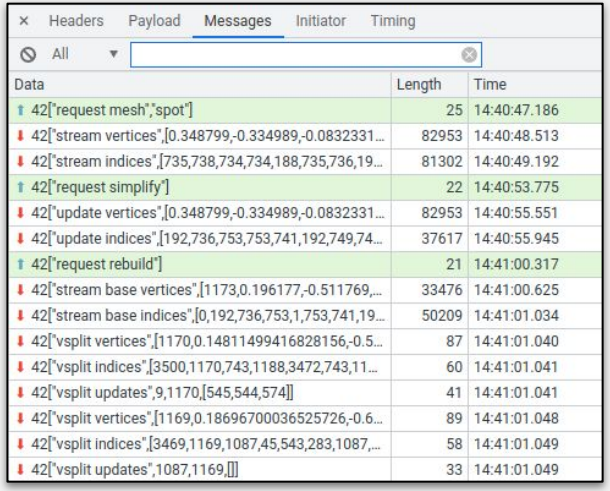
Progressive LOD Mesh Streaming with Three.js





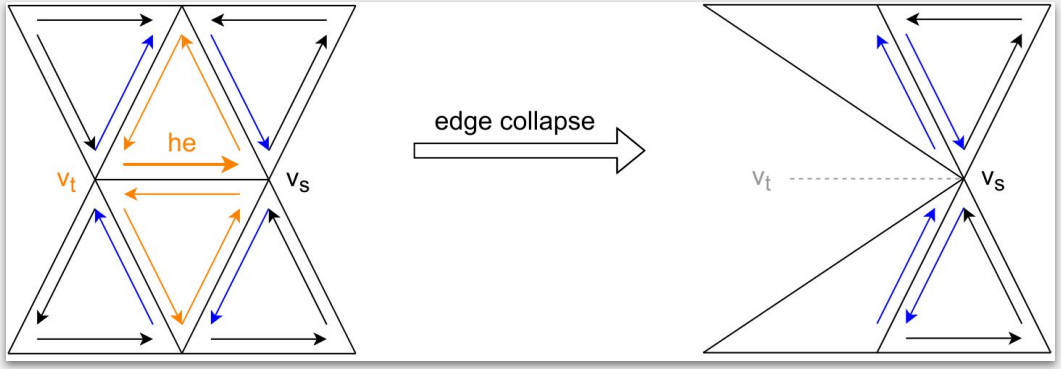
Progressive LOD Mesh Streaming with Three.js

Original Mesh



Data	Length	Time
42["request mesh","spot"]	25	14:40:47.186
42["stream vertices",[0.348799,-0.334989,-0.0832331...	82953	14:40:48.513
42["stream indices",[735,738,734,734,188,735,736,19...	81302	14:40:49.192
42["request simplify"]	22	14:40:53.775
42["update vertices",[0.348799,-0.334989,-0.0832331...	82953	14:40:55.551
42["update indices",[192,736,753,753,741,192,749,74...	37617	14:40:55.945
42["request rebuild"]	21	14:41:00.317
42["stream base vertices",[1173,0.196177,-0.511769,...	33476	14:41:00.625
42["stream base indices",[0,192,736,753,1,753,741,19...	50209	14:41:01.034
42["vsplit vertices",[1170,0.14811499416828156,-0.5...	87	14:41:01.040
42["vsplit indices",[3500,1170,743,1188,3472,743,11...	60	14:41:01.041
42["vsplit updates",9,1170,[545,544,574]]	41	14:41:01.041
42["vsplit vertices",[1169,0.18696700036525726,-0.6...	89	14:41:01.048
42["vsplit indices",[3469,1169,1087,45,543,283,1087,...	58	14:41:01.049
42["vsplit updates",1087,1169,[]]	33	14:41:01.049

Original Mesh



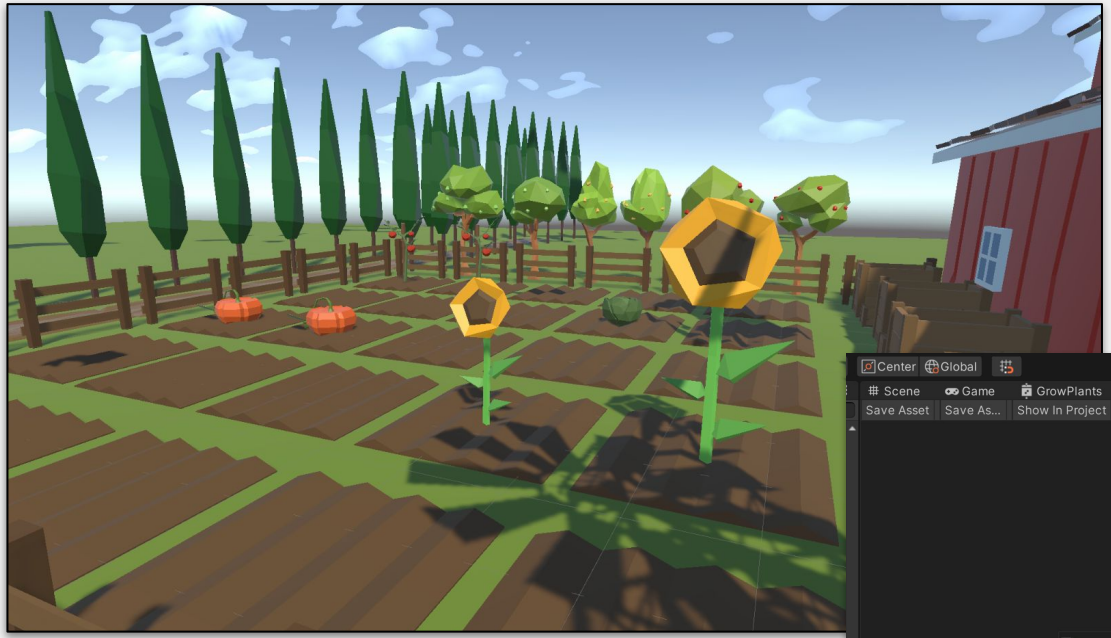
Progressive LOD Mesh Streaming with Three.js

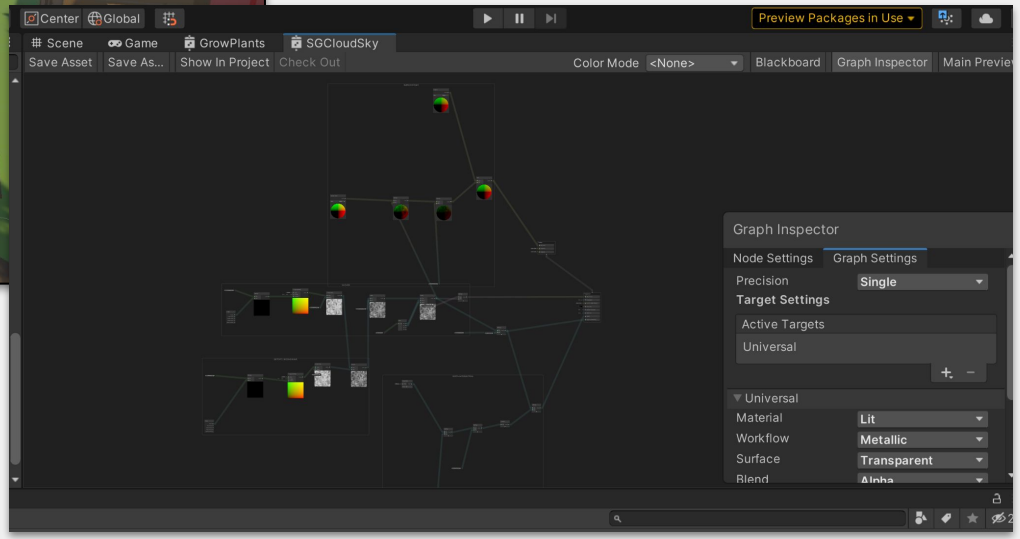
```
  h_ecolError(h: Halfedge) {
    //error of collapsing halfedge this---h.next.vert
    //two faces on either side of the halfedge
    //select face with biggest distance from those two faces
    let heLen = h.vector().norm();
    let change = 0;
    let incidentFaces = [];

    incidentFaces.push(h!.face);
    incidentFaces.push(h!.twin!.face);

    let max = 0;
    this.halfedges(he => {
      h.next!.vert.halfedges(he2 => {
        if(he.next!.vert!.idx === he2.next!.vert!.idx) {
          max++;
        }
      })
    })

    this.faces(f => {
      if(f.idx !== incidentFaces[0].idx && f.idx !== incidentFaces[1].idx) {
        let minChange = 1;
        incidentFaces.forEach(i => {
          minChange = Math.min(minChange, (1 - (f.normal().dot(i.normal())))) / 2);
        });
      change = Math.max(change, minChange);
      }
    });

    if(max !== 2) {
      this.manifold = false;
    }

    return heLen * change;
  }
}
```

Progressive LOD Mesh Streaming with Three.js

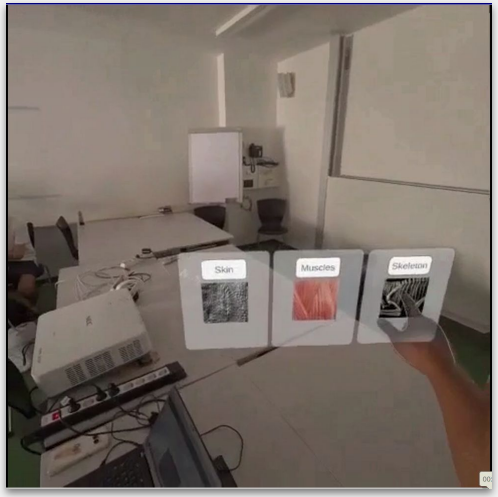
```
    lowest_ecolError() {
        let lowest: Vertex
        this.verts.forEach(v => {
            if(!v.rm && v.manifold && v.ecolProspect && !v.ecolProspect.rm && !lowest) {
                lowest = v;
            }
        });

        if(lowest) {
            this.verts.forEach(v => {
                if(!v.rm && v.manifold && v.ecolProspect && !v.ecolProspect.rm && v.ecolError < lowest.ecolError) {
                    lowest = v;
                }
            });
        }

        return lowest;
    }
}
```

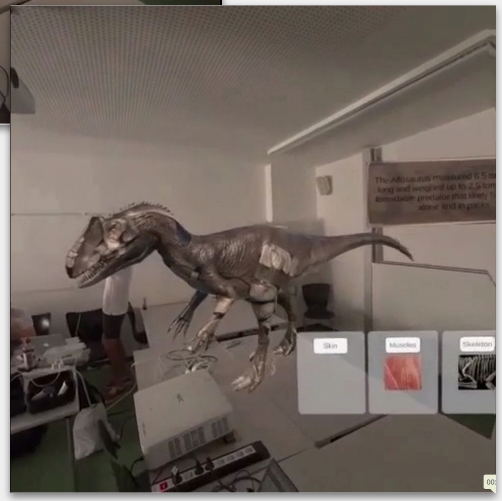
VR Farming Game with Unity



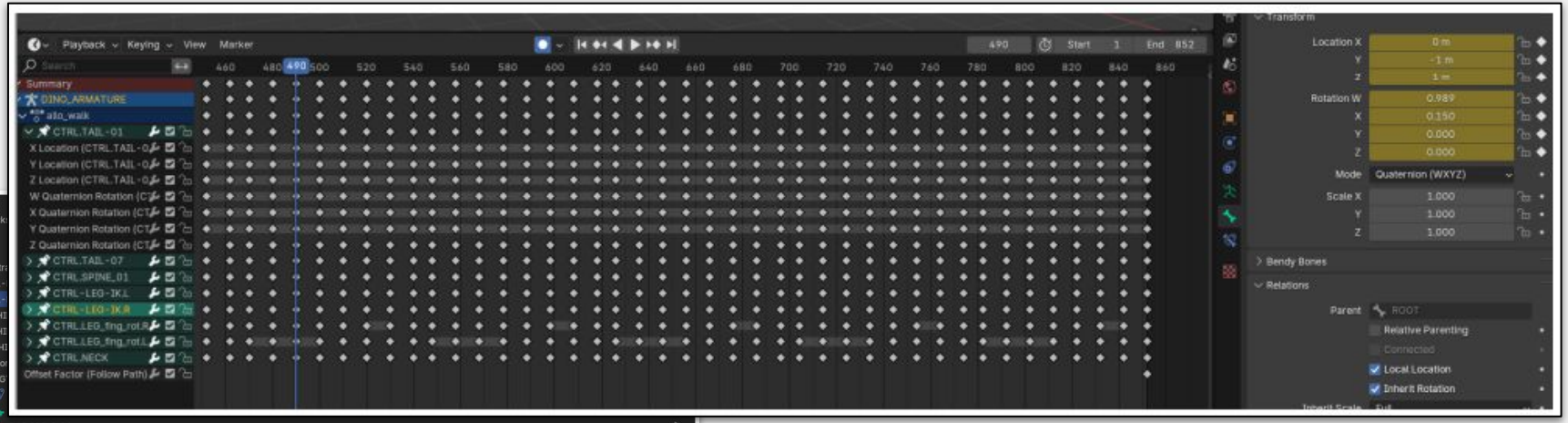


Hackathon - AR Museum App

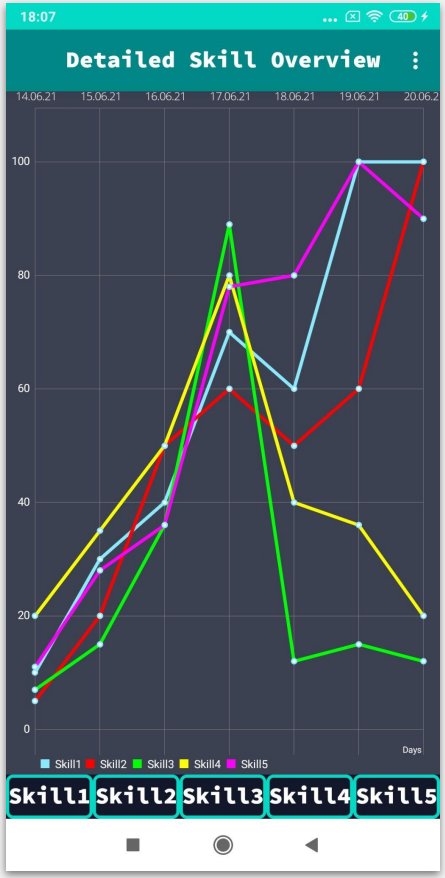


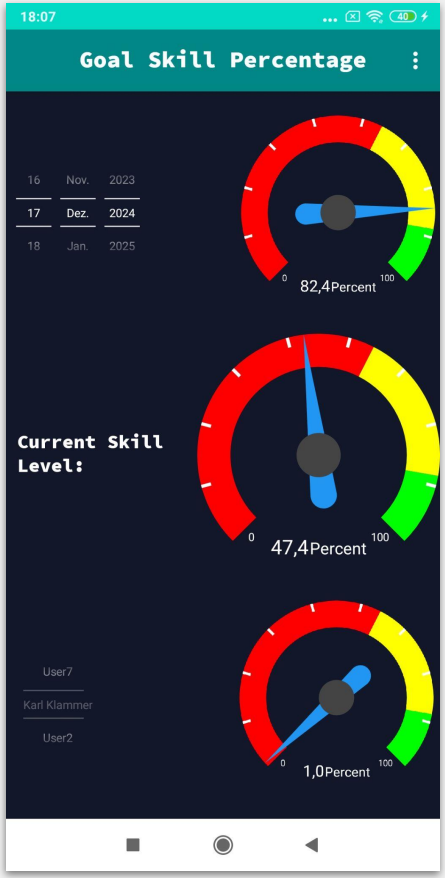


AR Museum App / Animations with Blender

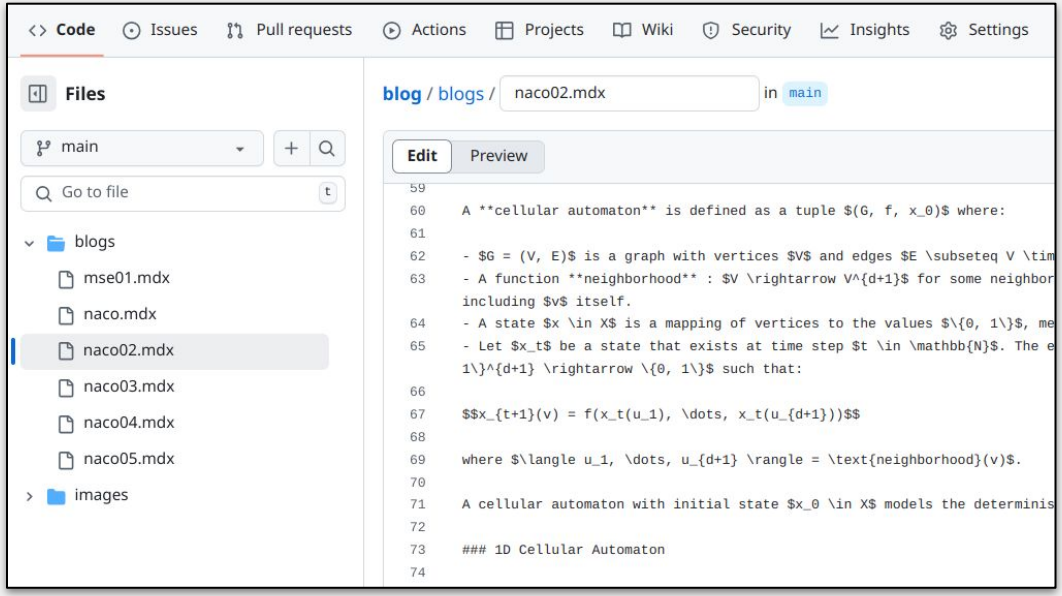


Android Study App





Learning Management System - MERN



A **cellular automaton** is defined as a tuple (G, f, x_0) where:

- $G = (V, E)$ is a graph with vertices V and edges $E \subseteq V \times V$.
- A function **neighborhood** : $V \rightarrow V^{d+1}$ for some neighborhood degree $d \in \mathbb{N}$, returns an ordered vector of neighbors of a given node v, always including v itself.
- A state $x \in X$ is a mapping of vertices to the values $\{0, 1\}$, meaning the state space X is defined as $X = (V \rightarrow \{0, 1\})$.
- Let x_t be a state that exists at time step $t \in \mathbb{N}$. The evolution of a state x_t to its subsequent state x_{t+1} is determined by a function $f : \{0, 1\}^{d+1} \rightarrow \{0, 1\}$ such that:

$$x_{t+1}(v) = f(x_t(u_1), \dots, x_t(u_{d+1}))$$

where $\langle u_1, \dots, u_{d+1} \rangle = \text{neighborhood}(v)$.

Learning Management System - MERN

Mutable vs. Immutable Data in OOP

Mutable Data

Objects with mutable state can change after they're created.

Example:

```
class Car(var doorState: String) {
  def openDoor(): Unit = {
    doorState = "OPEN"
  }
}

val car = new Car("CLOSED")
car.openDoor()
```

Considerations:

- **Real World**: Objects change state naturally (e.g., doors open)

Question 3 / 10:

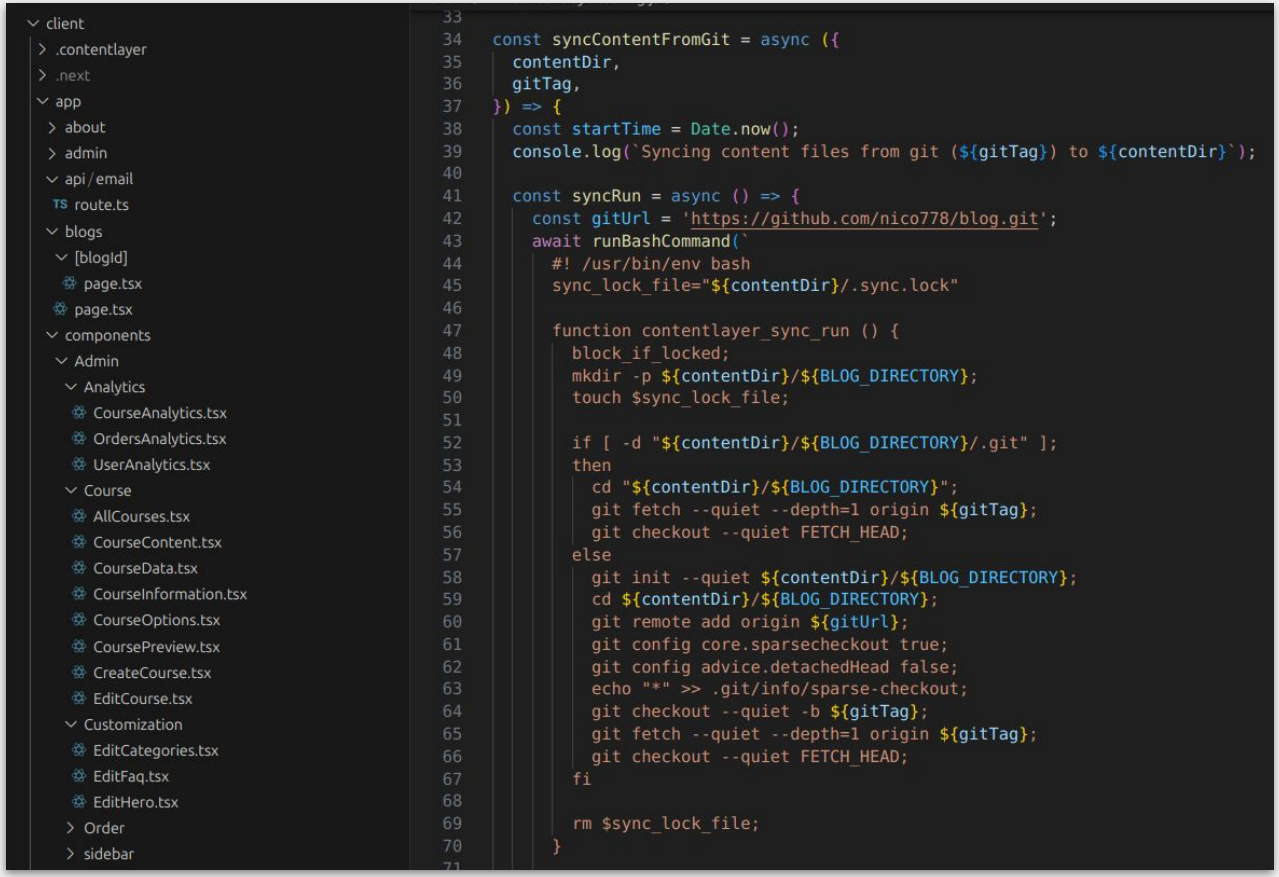
Which of the following best describes a 'Fault' in software testing? (10 marks)

Single Selection | Pick 1

- A programming or design flaw causing the system not to conform to its specification.
- A human mistake that leads to a problem in the system.
- A failure when the software is executed under certain conditions.
- An observable deviation from the specification.

Next

Learning Management System - MERN

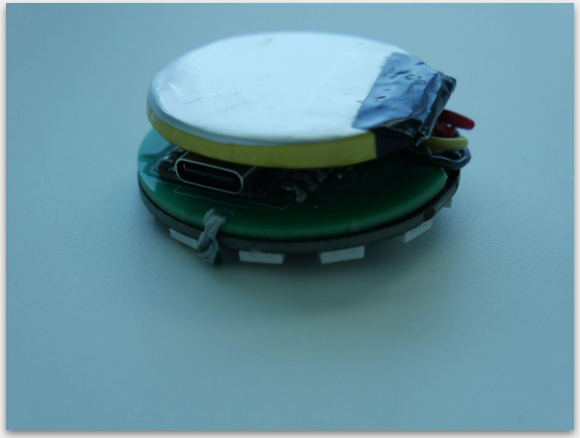


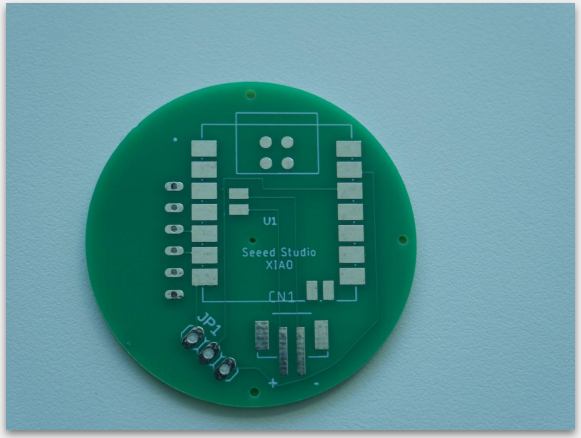
Learning Management System - MERN

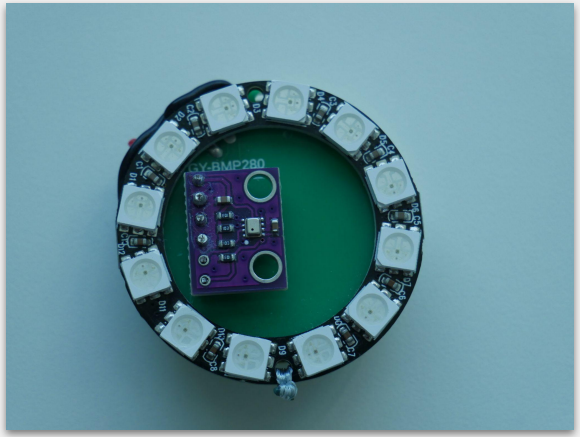
```tsx
const BlogDetails: React.FC<BlogDetailsProps> = ({ params }) => {
  const [post, setPost] = useState<Post | null>(null);
  const [quizData, setQuizData] = useState<any>(null);
  const [surveyData, setSurveyData] = useState<any>(null);

  useEffect(() => {
    const foundPost = allPosts.find((p: Post) => p.slug === params.blogId);
    if (!foundPost) {
      console.log('Post not found');
    } else {
      setPost(foundPost);

      // Extract quiz data from post.body.raw
      const quizJson = extractQuizData(foundPost.body.raw, 'json');
      if (quizJson) {
        try {
          const parsedQuiz = JSON.parse(quizJson);
          setQuizData(parsedQuiz);
        } catch (error) {
          console.error('Invalid JSON in quiz data:', error);
        }
      }

      const surveyJson = extractQuizData(foundPost.body.raw, 'surveyjson');
      if (surveyJson) {
        try {
          const parsedSurvey = JSON.parse(surveyJson);
          setSurveyData(parsedSurvey);
        } catch (error) {
          console.error('Invalid JSON in survey data:', error);
        }
      }
    }
  }, [params.blogId]);
```

```tsx
function extractQuizData(rawContent: string, codeBlockIdentifier: string): string | null {
  const regex = new RegExp('````' + codeBlockIdentifier + '([\\s\\S]*?)````', 'm');
  const match = rawContent.match(regex);
  if (match && match[1]) {
    return match[1].trim();
  }
  return null;
}

function removeQuizData(code: string): string {
  // Use a regular expression to remove the ```json ... ``` block
  const regex = /```json([\s\S]*?)```/;
  return code.replace(regex, '');
}
```

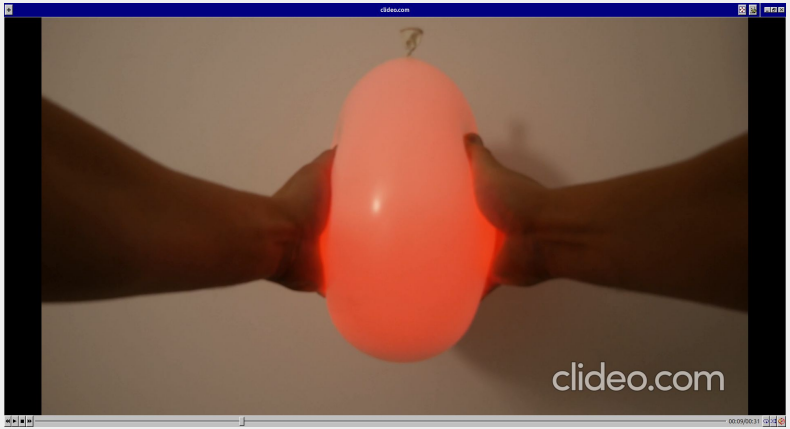
Smart Balloons - Developing HCI Prototypes



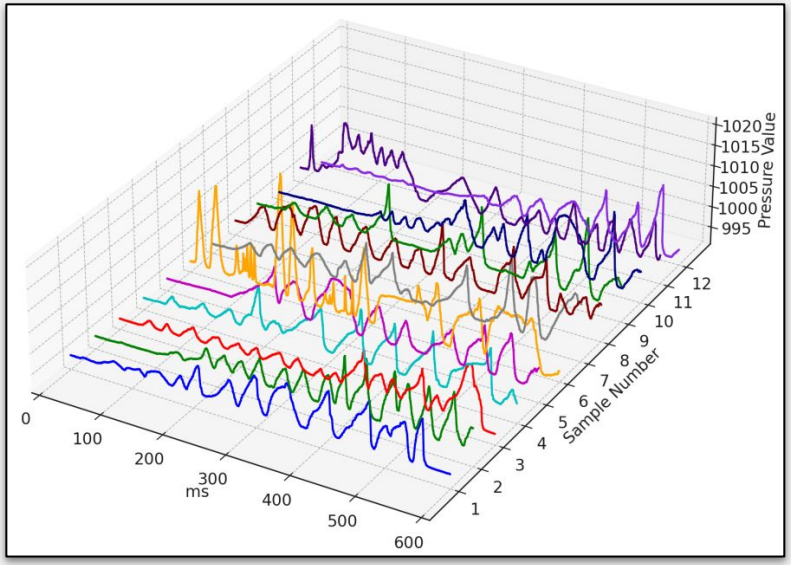




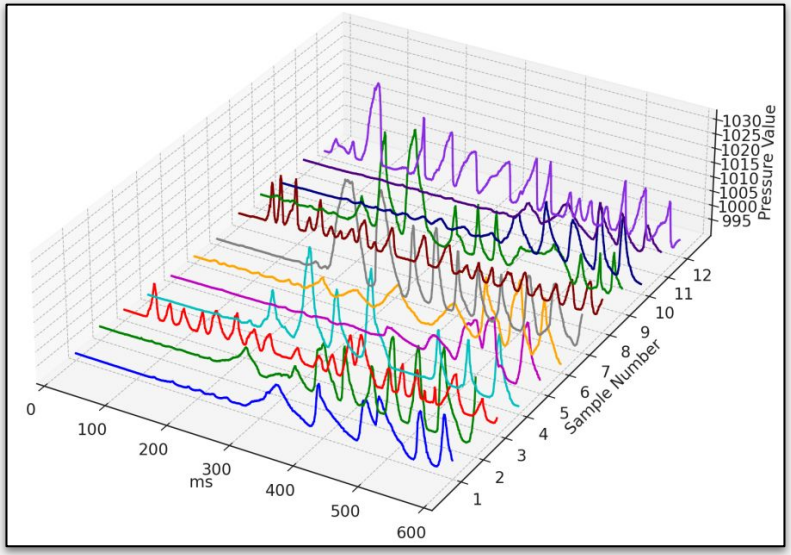
Smart Balloons - Developing HCI Prototypes



Evaluating Effects of LED Feedback on Interaction

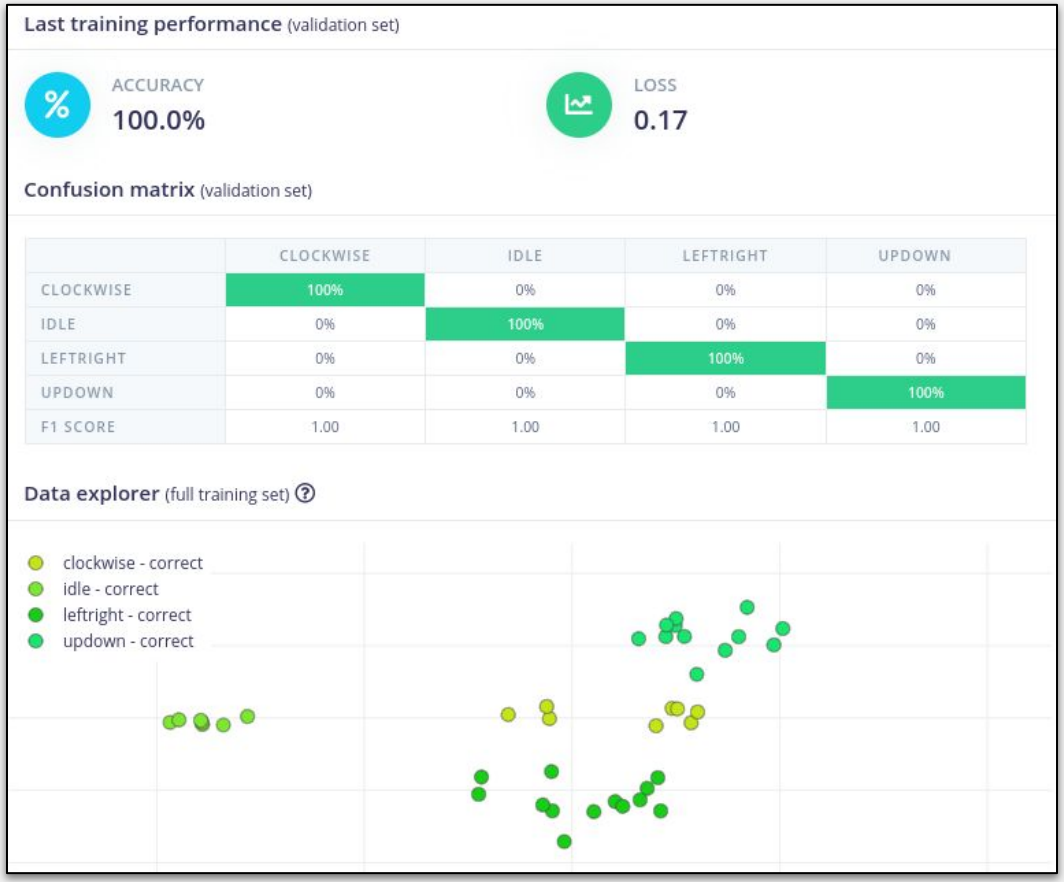


Green to Red



Red to Green

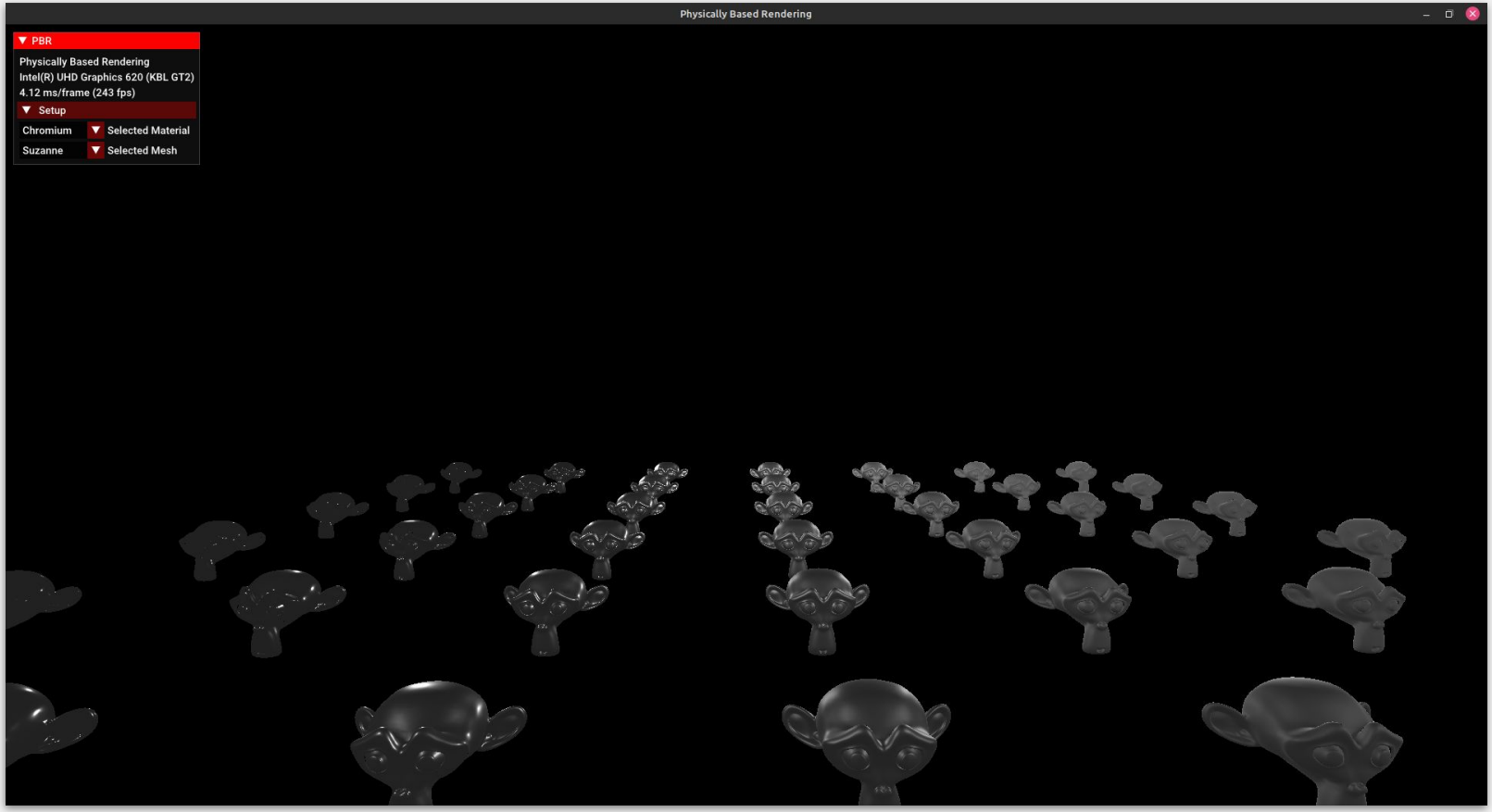
Smart Balloons - Gesture Recognition



Circuitpython - Proximity Example

```
scanning_time = random.uniform(MIN_SCANNING_TIME, MAX_SCANNING_TIME)
        for adv in ble.start_scan(timeout=scanning_time):
            try:
                device_name = adv.complete_name
                if device_name in TARGET_NAMES:
                    rssi_values_dict[device_name].append(adv.rssi)
```

Physically Based Rendering with Vulkan



Physically Based Rendering with Vulkan

```
vec3 BRDF(vec3 L, vec3 V, vec3 N, float metallic, float roughness)
{
    //Precalculate vectors and dot products
    vec3 H = normalize (V + L);
    float dotNV = clamp(dot(N, V), 0.0, 1.0);
    float dotNL = clamp(dot(N, L), 0.0, 1.0);
    float dotLH = clamp(dot(L, H), 0.0, 1.0);
    float dotNH = clamp(dot(N, H), 0.0, 1.0);

    //Light color fixed
    vec3 lightColor = vec3(1.0);

    vec3 color = vec3(0.0);

    if (dotNL > 0.0)
    {
        float rroughness = max(0.05, roughness);
        //D = Normal distribution (Distribution of the microfacets)
        float D = D_GGX(dotNH, roughness);
        //G = Geometric shadowing term (Microfacets shadowing)
        float G = G_SchlicksmithGGX(dotNL, dotNV, rroughness);
        //F = Fresnel factor (Reflectance depending on angle of incidence)
        vec3 F = F_Schlick(dotNV, metallic);

        vec3 spec = D * F * G / (4.0 * dotNL * dotNV);

        color += spec * dotNL * lightColor;
    }

    return color;
}
```

Physically Based Rendering with Vulkan

```
VK_CHECK_RESULT(vkBeginCommandBuffer(drawCmdBuffers[i], &cmdBufInfo));

vkCmdBeginRenderPass(drawCmdBuffers[i], &rPB_Info, VK_SUBPASS_CONTENTS_INLINE);

VkViewport vp = vks::initializers::viewport((float)width, (float)height, 0.0f, 1.0f);
vkCmdSetViewport(drawCmdBuffers[i], 0, 1, &vp);

VkRect2D scis = vks::initializers::rect2D(width, height, 0, 0);
vkCmdSetScissor(drawCmdBuffers[i], 0, 1, &scis);

//objects
vkCmdBindPipeline(drawCmdBuffers[i], VK_PIPELINE_BIND_POINT_GRAPHICS, pl);
vkCmdBindDescriptorSets(drawCmdBuffers[i], VK_PIPELINE_BIND_POINT_GRAPHICS, pl_Layout, 0, 1, &dSet, 0, NULL);

Material material = materials[material_ID];

//draw materials
for (uint32_t y = 0; y < FIELD; y++) {
    for (uint32_t x = 0; x < FIELD; x++) {
        glm::vec3 position = glm::vec3(float(x - (FIELD / 2.0f)) * 2.5f, 0.0f, float(y - (FIELD / 2.0f)) * 2.5f);
        vkCmdPushConstants(drawCmdBuffers[i], pl_Layout, VK_SHADER_STAGE_VERTEX_BIT, 0, sizeof(glm::vec3), &position);
        material.props.metallic = glm::clamp((float)x / (float)(FIELD - 1), 0.1f, 1.0f);
        material.props.roughness = glm::clamp((float)y / (float)(FIELD - 1), 0.05f, 1.0f);
        vkCmdPushConstants(drawCmdBuffers[i], pl_Layout, VK_SHADER_STAGE_FRAGMENT_BIT, sizeof(glm::vec3), sizeof(Material::VulkanPC), &material);
        meshes.artefacts[meshes.artefactID].draw(drawCmdBuffers[i]);
    }
}

drawUI(drawCmdBuffers[i]);

vkCmdEndRenderPass(drawCmdBuffers[i]);

VK_CHECK_RESULT(vkEndCommandBuffer(drawCmdBuffers[i]));
```