Treball de Fi de Grau

Enginyeria en Tecnologies Industrials

Dispositius hàptics aplicats a la interacció amb imatges mèdiques de volum

ANNEXOS DE PROGRAMACIÓ

Autor: Magalí May Llorens
Director: Toni Susin Sanchez

Convocatòria: Juny 2018

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum contingut

En aquest document hi ha els *scripts* que més s'han utilitzat per a la realització del treball de fi de grau de "*Dispositius hàptics aplicats a la interacció amb imatges mèdiques de volum*". La resta d'annexos estan juntament amb la memòria.

Índex

Script Menu_Haptic

F:\VRMedCode\Menu_Haptic.cs 1

```
using UnityEngine;
using System.Collections;
using System.IO;
using UnityEngine.EventSystems;
using System.Collections.Generic;
using UnityEngine.UI;
using System.Linq;
using System.Runtime.InteropServices;
using UnityEngine.SceneManagement;

public class Menu_Haptic : MonoBehaviour {
    public GameObject volume;
    public VolumeRenderer vr;
    bool raycastDone;
    public static string currentState = "Move";
    public GameObject Sphere;
    TransferFunction tf;
    //Haptic
    public Text Data;
    public GameObject PaintedObjects;
    public GameObject dummy;
    public GameObject dummy2;
    float updateTime;
    public GameObject Canvas;

    //Rotation with Haptic
    //PAINT
    bool startPaint;

    TFRange range;
    float lastintensity;
    // haptiquem
    string nomob;
    public GameObject Cercle;
    //canvas info
    GameObject canvasinfo;
    GameObject canvasinfo2;
    GameObject canvasseleccio;
    GameObject menu;
    int pos;
    string pastState = "Move";
    //quina escena estem
    string nomscen;
    Scene scene;

    int rangeN = 0;
    // Use this for initialization
    void Start()
    {

        nomob = "";
        canvasinfo=GameObject.Find("Canvasinformatiu");
        canvasinfo.SetActive (false);
        canvasinfo2=GameObject.Find("Canvasinformatiu2");
        canvasinfo2.SetActive (false);
        canvasseleccio=GameObject.Find("Canvasseleccio");
```

```
57          canvasseleccio.SetActive (false);
58          menu = GameObject.Find ("Menuselec");
59          GameObject.Find ("PaintedVoxels").transform.rotation = GameObject.Find
               ("Cube").transform.rotation;
60          //maneig scenes
61          scene=SceneManager.GetActiveScene();
62          nomscen = scene.name;
63
64          tf = volume.GetComponent<VolumeRenderer>().transferFunction;
65
66          //Paint
67          range = new TFRange();
68          range.color = Color.green;
69          vr.transferFunction.ranges.Add(range);
70          vr.transferFunction.ranges[5].hide = true;
71          vr.transferFunction.UpdateTexture();
72
73      }
74
75      // Update is called once per frame
76      void LateUpdate()
77      {
78          if (Input.GetKeyDown(KeyCode.Space)) {
79              UnityEngine.SceneManagement.SceneManager.LoadScene("main");
80              scene=SceneManager.GetActiveScene();
81              nomscen = scene.name;
82
83          }
84          if (PluginImport.GetContact ()) {
85              nomob = ConverterClass.ConvertIntPtrToByteToString
                   (PluginImport.GetTouchedObjectName ());
86
87              if (nomob == "PlaneM") {
88                  Colorscorrectes ();
89                  GameObject.Find ("M").GetComponent<SpriteRenderer> ().color =
                       new Color (1, 0.66f, 0);
90              } else if (nomob == "PlaneC") {
91                  Colorscorrectes ();
92                  GameObject.Find ("C").GetComponent<SpriteRenderer> ().color =
                       new Color (1, 0.66f, 0);
93              } else if (nomob == "PlaneA") {
94                  Colorscorrectes ();
95                  GameObject.Find ("A").GetComponent<SpriteRenderer> ().color =
                       new Color (1, 0.66f, 0);
96              } else if (nomob == "PlaneF") {
97                  Colorscorrectes ();
98                  GameObject.Find ("F").GetComponent<SpriteRenderer> ().color =
                       new Color (1, 0.66f, 0);
99              } else if (nomob == "PlaneT") {
100                 GameObject.Find ("T").GetComponent<SpriteRenderer> ().color =
                       new Color (1, 0.66f, 0);
101             } else if (nomob == "PlaneP") {
102                 Colorscorrectes ();
103                 GameObject.Find ("P").GetComponent<SpriteRenderer> ().color =
                       new Color (1, 0.66f, 0);
104             } else if (nomob == "PlaneAV") {
```

```
105                 Colorscorrectes ();
106                 GameObject.Find ("AV").GetComponent<SpriteRenderer> ().color =
                        new Color (1, 0.66f, 0);
107             } else if (nomob == "PlaneInst") {
108                 Colorscorrectes ();
109                 GameObject.Find ("Inst").GetComponent<SpriteRenderer> ().color
                        = new Color (1, 0.66f, 0);
110             } else if (nomob == "PlaneSp") {
111                 Colorscorrectes ();
112                 GameObject.Find ("Sp").GetComponent<SpriteRenderer> ().color =
                        new Color (1, 0.66f, 0);
113             } else if (nomob == "PlaneNH") {
114                 Colorscorrectes ();
115                 GameObject.Find("NH").GetComponent<SpriteRenderer>().color =
                      new Color(1, 0.66f, 0);
116             }
117
118             if (PluginImport.GetButtonState(1, 1)||Input.GetKeyDown
                  (KeyCode.A)) {
119
120                 if (nomob == "PlaneM") {
121                     EnterMoveState ();
122                 } else if (nomob == "PlaneC") {
123                     EnterCropState ();
124                 } else if (nomob == "PlaneA") {
125                     EnterAnalisisState ();
126                 } else if (nomob == "PlaneF") {
127                     EnterForceState ();
128                 } else if (nomob == "PlaneT") {
129                     EnterTouchState ();
130                 } else if (nomob == "PlaneP") {
131                     EnterPaintState ();
132                 } else if (nomob == "PlaneAV") {
133                     EnterAVibState ();
134                 } else if (nomob == "PlaneInst" && canvasseleccio.activeSelf
                      == false&&canvasinfo2.activeSelf == false) {
135                     currentState = "Instruccions";
136                     Colorscorrectes ();
137                     canvasinfo.SetActive (true);
138                 } else if (nomob == "PlaneSp" && canvasinfo.activeSelf ==
                      false&&canvasinfo2.activeSelf == false) {
139                     currentState = "Sp";
140                     Colorscorrectes ();
141                     canvasseleccio.SetActive (true);
142                 } else if (nomob == "PlaneNH") {
143                     currentState = "NH";
144                     Colorscorrectes ();
145                     UnityEngine.SceneManagement.SceneManager.LoadScene
                        ("main");
146                     scene = SceneManager.GetActiveScene ();
147                     nomscen = scene.name;
148                 }
149
150
151             }
152
```

```
153         }
154         /*if (UnityEngine.SceneManager.GetActiveScene ()) {
155 
156         }*/
157 
158 
159         if (PluginImport.GetContact()==false) {
160             Colorscorrectes ();
161         }
162         if (Input.GetKeyDown (KeyCode.Alpha9)
              &&canvasseleccio.activeSelf==false) {
163             currentState = "Instruccions";
164             Colorscorrectes ();
165             Tancainfo ();
166         }
167         if (Input.GetKeyDown (KeyCode.Alpha8)
              &&canvasinfo.activeSelf==false&&canvasinfo2.activeSelf==false) {
168             currentState = "Sp";
169             Colorscorrectes ();
170             Tancaselec ();
171         }
172         if ((canvasinfo.activeSelf||canvasinfo2.activeSelf)&&
              (PluginImport.GetButton2State ()||Input.GetKey(KeyCode.B))) {
173             Tancainfo();
174             currentState = pastState;
175         }
176         if (canvasseleccio.activeSelf&&(PluginImport.GetButton2State ()||
              Input.GetKey(KeyCode.B))) {
177             canvasseleccio.SetActive (false);
178             currentState = pastState;
179         }
180         if(canvasseleccio.activeSelf==false&&currentState=="Sp"){
181             Entrardespresdetancarmenu ();
182         }
183         if
              (canvasinfo.activeSelf==false&&canvasinfo2.activeSelf==false&&curren
              tState=="Instruccions"){
184             Entrardespresdetancarmenu();
185         }
186         if ((canvasinfo.activeSelf == true && Input.GetKeyDown
              (KeyCode.RightArrow))||(canvasinfo2.activeSelf == true &&
              Input.GetKeyDown (KeyCode.LeftArrow))) {
187             next ();
188         }
189         /*if (Input.GetKey(KeyCode.D)){
190             vr.transferFunction.ranges[1].hide = true;
191         }*/
192         if (Input.GetKeyDown(KeyCode.Alpha1)) {
193 
194             EnterMoveState();
195 
196         }else if (Input.GetKeyDown(KeyCode.Alpha2))
197         {
198 
199             EnterCropState();
200         }
```

```
201         else if (Input.GetKeyDown(KeyCode.Alpha3))
202         {
203
204             EnterAnalisisState();
205         }
206         else if (Input.GetKeyDown(KeyCode.Alpha4))
207         {
208             EnterForceState();
209         }
210         else if (Input.GetKeyDown(KeyCode.Alpha5))
211         {
212             EnterTouchState();
213         }
214         else if (Input.GetKeyDown(KeyCode.Alpha6))
215         {
216             EnterPaintState();
217         }
218         else if (Input.GetKeyDown(KeyCode.Alpha7))
219         {
220             EnterAVibState();
221         }
222         //tf.ChangeTexture (GameObject.Find
              ("canvicolors1").GetComponent<SpriteRenderer> ().color, rangeN);
223         #region AVib
224         if (currentState == "AVib")
225         {
226             updateTime += Time.deltaTime;
227             if (updateTime >= 0.3f)
228             {
229                 updateTime = 0;
230
231                 float[] directionEffect = new float[3];
232
233                 directionEffect[0] = 0;
234                 directionEffect[1] = 0;
235                 directionEffect[2] = 0;
236
237
238                 float intensity = vr.SampleVolume(Sphere.transform.position);
239                 lastintensity = 3 * lastintensity + intensity;
240                 lastintensity /= 4;
241                 Data.text = (intensity * 100).ToString("00");
242
243                 //Set the effect
244                 /*PluginImport.SetEffect
                      (ConverterClass.ConvertStringToByteToIntPtr
                      ("vibrationMotor"), 1, 0, Mathf.Clamp(intensity*2,0,1), 0,
                      10+40*intensity, ConverterClass.ConvertFloat3ToIntPtr
                      (directionEffect), ConverterClass.ConvertFloat3ToIntPtr
                      (directionEffect));
245                 PluginImport.StartEffect(1);*/
246                 PluginImport.SetEffect
                      (ConverterClass.ConvertStringToByteToIntPtr ("friction"), 5,
                       Mathf.Clamp(intensity*2,0,1), Mathf.Clamp(intensity*2,0,1),
                       300f, 0, ConverterClass.ConvertFloat3ToIntPtr
                      (directionEffect), ConverterClass.ConvertFloat3ToIntPtr
```

```
                    (directionEffect));
247                 PluginImport.StartEffect (5);
248
249             }
250         }
251         #endregion
252
253         #region Analisi
254
255         if (currentState == "Analisis")
256         {
257             if (PluginImport.GetButton1State()||Input.GetKey(KeyCode.A))
258             {
259
260                 if (!startPaint)
261                 {
262                     startPaint = true;
263
264
265                     for (int i = 0; i< 4; i++){
266                         vr.transferFunction.ranges[i].hide = true;
267                     }
268
269                     vr.transferFunction.ranges[4].hide = false;
270                     vr.transferFunction.ranges[5].hide = false;
271                 }
272
273                 else
274                 {
275                     float intensity = vr.SampleVolume
                    (Sphere.transform.position);
276                     lastintensity = 3 * lastintensity + intensity;
277                     lastintensity /= 4;
278
279                     Data.text = (intensity * 100).ToString("00");
280                     range.listCordenadas[0] = new Vector2(lastintensity-0.15f,
                     0);
281                     range.listCordenadas[1] = new Vector2(lastintensity -
                    0.1f, 1);
282                     range.listCordenadas[2] = new Vector2(lastintensity +
                    0.1f, 1);
283                     range.listCordenadas[3] = new Vector2(lastintensity+0.15f,
                     0);
284
285                     vr.transferFunction.UpdateTexture();
286                 }
287             }
288             else
289             {
290                 startPaint = false;
291
292                 for (int i = 0; i < 4; i++)
293                 {
294                     vr.transferFunction.ranges[i].hide = false;
295                 }
296
```

```
297             }
298         }
299         #endregion
300
301         #region Paint
302
303         if (currentState == "Paint") {
304
305             if (PluginImport.GetButton1State()||Input.GetKey(KeyCode.A))
306             {
307                 GameObject PaintVox = (GameObject)Instantiate
                      (Resources.Load<GameObject>("PaintVox"),
                      Sphere.transform.position, Quaternion.identity,
                      PaintedObjects.transform);
308
309                 PaintVox.transform.localScale = Vector3.one * 0.01f /
                      PaintedObjects.transform.localScale.x;
310
311             }
312             if (Input.GetKey(KeyCode.C))
313             {
314                 if (vr.Pint(Sphere.transform.position)){
315                     GameObject PaintVox = (GameObject)Instantiate
                          (Resources.Load<GameObject>("PaintVox2"),
                          Sphere.transform.position, Quaternion.identity,
                          PaintedObjects.transform);
316
317                     PaintVox.transform.localScale = Vector3.one * 0.01f /
                          PaintedObjects.transform.localScale.x;
318                 }
319
320             }
321
322             else if (PluginImport.GetButton2State()||Input.GetKey(KeyCode.B))
                  {
323                 nomob = ConverterClass.ConvertIntPtrToByteToString
                      (PluginImport.GetTouchedObjectName ());
324
325                 Transform[] childs =
                      PaintedObjects.transform.GetComponentsInChildren<Transform>
                      ();
326
327                 for (int i = childs.Length-1; i >0; i--) {
328                     if ((Sphere.transform.position - childs
                          [i].transform.position).magnitude < 0.05f) {
329                         Destroy(childs[i].gameObject);
330                     }
331
332
333                 }
334             }
335         }
336         #endregion
337
338     }
339
```

```
340
341
342     public void EnterMoveState()
343     {
344
345         canviestatdespresanalisis ();
346         currentState = "Move";
347         pastState = currentState;
348         Colorscorrectes();
349         GameObject.Find("Selector").transform.position = GameObject.Find
              ("Move").transform.position;
350
351         GameObject.Find("Cube").transform.position =
              volume.transform.TransformPoint(volume.GetComponent<VolumeRenderer>
              ().bounds.center);
352         //GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().enabled = true;
353         GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().EndGrab();
354         PluginImport.StopEffect(1);
355         if (!dummy.activeSelf) {
356             PluginImport.HapticCleanUp();
357             dummy2.SetActive(false);
358             dummy.SetActive(true);
359             dummy.GetComponent<SimpleShapeContact>().Start();
360         }
361     }
362     public void EnterCropState()
363     {
364
365         canviestatdespresanalisis ();
366         currentState = "Crop";
367         pastState = currentState;
368         Colorscorrectes();
369         GameObject.Find("Selector").transform.position = GameObject.Find
              ("Crop").transform.position;
370
371         GameObject.Find("Cube").transform.position =
              volume.transform.TransformPoint(volume.GetComponent<VolumeRenderer>
              ().bounds.center);
372         //GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().enabled = true;
373         GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().EndGrab();
374         PluginImport.StopEffect(1);
375         if (!dummy.activeSelf)
376         {
377             PluginImport.HapticCleanUp();
378             dummy2.SetActive(false);
379             dummy.SetActive(true);
380             dummy.GetComponent<SimpleShapeContact>().Start();
381         }
382     }
383     public void EnterAVibState()
384     {
385
```

```
386         canviestatdespresanalisis ();
387         currentState = "AVib";
388         pastState = currentState;
389         Colorscorrectes();
390         GameObject.Find("Selector").transform.position = GameObject.Find
              (currentState).transform.position;
391         GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().EndGrab();
392         GameObject.Find("Cube").transform.position = Vector3.forward * 10;
393         PluginImport.StopEffect(1);
394         PluginImport.HapticCleanUp();
395         dummy.SetActive(false);
396         dummy2.SetActive(true);
397         dummy.GetComponent<SimpleShapeContact>().Start();
398
399
400         //PluginImport.SetMode(2);
401
402
403     }
404     public void EnterForceState()
405     {
406
407         canviestatdespresanalisis ();
408         currentState = "Force";
409         pastState = currentState;
410         Colorscorrectes();
411         GameObject.Find("Selector").transform.position = GameObject.Find
              ("Force").transform.position;
412         GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().EndGrab();
413         //GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().enabled = false;
414         GameObject.Find("Cube").transform.position = Vector3.forward * 10;
415         PluginImport.StopEffect(1);
416         //UpdateHapticObjects =true;
417
418         if (!dummy.activeSelf)
419         {
420             PluginImport.HapticCleanUp();
421             dummy2.SetActive(false);
422             dummy.SetActive(true);
423             dummy.GetComponent<SimpleShapeContact>().Start();
424         }
425
426     }
427     public void EnterTouchState()
428     {
429
430         canviestatdespresanalisis ();
431         currentState = "Touch";
432         pastState = currentState;
433         Colorscorrectes();
434         GameObject.Find("Selector").transform.position = GameObject.Find
              ("Touch").transform.position;
435         GameObject.Find("Crop Range
```

```
            Editor").GetComponent<CropRangeEditor_Haptic>().EndGrab();
436         //GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().enabled = false;
437         GameObject.Find("Cube").transform.position = Vector3.forward * 10;
438         PluginImport.StopEffect(1);
439         //UpdateHapticObjects =true;
440
441         if (!dummy.activeSelf)
442         {
443             PluginImport.HapticCleanUp();
444             dummy2.SetActive(false);
445             dummy.SetActive(true);
446             dummy.GetComponent<SimpleShapeContact>().Start();
447         }
448
449     }
450     public void EnterPaintState()
451     {
452
453         canviestatdespresanalisis ();
454         currentState = "Paint";
455         pastState = currentState;
456         Colorscorrectes();
457         GameObject.Find("Selector").transform.position = GameObject.Find
              (currentState).transform.position;
458         GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().EndGrab();
459         GameObject.Find("Cube").transform.position = Vector3.forward * 10;
460         PluginImport.StopEffect(1);
461         if (!dummy.activeSelf)
462         {
463             PluginImport.HapticCleanUp();
464             dummy2.SetActive(false);
465             dummy.SetActive(true);
466             dummy.GetComponent<SimpleShapeContact>().Start();
467         }
468     }
469     public void EnterAnalisisState()
470     {
471
472         canviestatdespresanalisis ();
473         currentState = "Analisis";
474         pastState = currentState;
475         Colorscorrectes();
476         GameObject.Find("Selector").transform.position = GameObject.Find
              (currentState).transform.position;
477         GameObject.Find("Crop Range
              Editor").GetComponent<CropRangeEditor_Haptic>().EndGrab();
478         GameObject.Find("Cube").transform.position = Vector3.forward * 10;
479         PluginImport.StopEffect(1);
480
481         if (!dummy.activeSelf)
482         {
483             PluginImport.HapticCleanUp();
484             dummy2.SetActive(false);
485             dummy.SetActive(true);
```

```
486             dummy.GetComponent<SimpleShapeContact>().Start();
487         }
488     }
489     void Colorscorrectes()
490     {
491         if (Sphere.transform.position.z + 0.05 < menu.transform.position.z) {
492             GameObject.Find ("interM").GetComponent<SpriteRenderer> ().color =
                  new Color (1, 1, 1, 1);
493             GameObject.Find ("interC").GetComponent<SpriteRenderer> ().color =
                  new Color (1, 1, 1, 1);
494             GameObject.Find ("interA").GetComponent<SpriteRenderer> ().color =
                  new Color (1, 1, 1, 1);
495             GameObject.Find ("interF").GetComponent<SpriteRenderer> ().color =
                  new Color (1, 1, 1, 1);
496             GameObject.Find ("interT").GetComponent<SpriteRenderer> ().color =
                  new Color (1, 1, 1, 1);
497             GameObject.Find ("interP").GetComponent<SpriteRenderer> ().color =
                  new Color (1, 1, 1, 1);
498             GameObject.Find ("interAV").GetComponent<SpriteRenderer> ().color
                  = new Color (1, 1, 1, 1);
499             GameObject.Find ("interInst").GetComponent<SpriteRenderer>
                  ().color = new Color (1, 1, 1, 1);
500             GameObject.Find ("interSp").GetComponent<SpriteRenderer> ().color
                  = new Color (1, 1, 1, 1);
501             GameObject.Find ("M").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 1);
502             GameObject.Find ("C").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 1);
503             GameObject.Find ("A").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 1);
504             GameObject.Find ("F").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 1);
505             GameObject.Find ("T").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 1);
506             GameObject.Find ("P").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 1);
507             GameObject.Find ("AV").GetComponent<SpriteRenderer> ().color = new
                   Color (0, 0, 0, 1);
508             GameObject.Find ("Inst").GetComponent<SpriteRenderer> ().color =
                  new Color (0, 0, 0, 1);
509             GameObject.Find ("Sp").GetComponent<SpriteRenderer> ().color = new
                   Color (0, 0, 0, 1);
510             GameObject.Find ("NH").GetComponent<SpriteRenderer> ().color = new
                   Color (1, 1, 0, 1);
511             GameObject.Find ("Cercle").GetComponent<SpriteRenderer> ().color =
                   new Color (0, 1, 0, 1);
512             if (currentState == "Move") {
513                 GameObject.Find ("M").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 1);
514                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneM").transform.position;
515             } else if (currentState == "Crop") {
516                 GameObject.Find ("C").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 1);
517                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneC").transform.position;
```

```
518             } else if (currentState == "Analisis") {
519                 GameObject.Find ("A").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 1);
520                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneA").transform.position;
521             } else if (currentState == "Force") {
522                 GameObject.Find ("F").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 1);
523                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneF").transform.position;
524             } else if (currentState == "Touch") {
525                 GameObject.Find ("T").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 1);
526                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneT").transform.position;
527             } else if (currentState == "Paint") {
528                 GameObject.Find ("P").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 1);
529                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneP").transform.position;
530             } else if (currentState == "AVib") {
531                 GameObject.Find ("AV").GetComponent<SpriteRenderer> ().color =
                       new Color (0, 1f, 0, 1);
532                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneAV").transform.position;
533             } else if (currentState == "Instruccions") {
534                 GameObject.Find ("Inst").GetComponent<SpriteRenderer> ().color
                       = new Color (0, 1f, 0, 1);
535                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneInst").transform.position;
536             } else if (currentState == "Sp") {
537                 GameObject.Find ("Sp").GetComponent<SpriteRenderer> ().color =
                       new Color (0, 1f, 0, 1);
538                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneSp").transform.position;
539             }
540         }
541         if(Sphere.transform.position.z+0.05>menu.transform.position.z){
542             GameObject.Find("interM").GetComponent<SpriteRenderer>().color =
                  new Color(1, 1, 1, 0.5f);
543             GameObject.Find("interC").GetComponent<SpriteRenderer>().color =
                  new Color(1, 1, 1, 0.5f);
544             GameObject.Find("interA").GetComponent<SpriteRenderer>().color =
                  new Color(1, 1, 1, 0.5f);
545             GameObject.Find("interF").GetComponent<SpriteRenderer>().color =
                  new Color(1, 1, 1, 0.5f);
546             GameObject.Find("interT").GetComponent<SpriteRenderer>().color =
                  new Color(1, 1, 1, 0.5f);
547             GameObject.Find("interP").GetComponent<SpriteRenderer>().color =
                  new Color(1, 1, 1, 0.5f);
548             GameObject.Find("interAV").GetComponent<SpriteRenderer>().color =
                  new Color(1, 1, 1, 0.5f);
549             GameObject.Find("interInst").GetComponent<SpriteRenderer>().color
                  = new Color(1, 1, 1, 0.5f);
550             GameObject.Find("interSp").GetComponent<SpriteRenderer>().color =
                  new Color(1, 1, 1, 0.5f);
```

```
551             GameObject.Find ("M").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 0.5f);
552             GameObject.Find ("C").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 0.5f);
553             GameObject.Find ("A").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 0.5f);
554             GameObject.Find ("F").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 0.5f);
555             GameObject.Find ("T").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 0.5f);
556             GameObject.Find ("P").GetComponent<SpriteRenderer> ().color = new
                  Color (0, 0, 0, 0.5f);
557             GameObject.Find ("AV").GetComponent<SpriteRenderer> ().color = new
                   Color (0, 0, 0, 0.5f);
558             GameObject.Find ("Inst").GetComponent<SpriteRenderer> ().color =
                  new Color (0, 0, 0, 0.5f);
559             GameObject.Find ("Sp").GetComponent<SpriteRenderer> ().color = new
                   Color (0, 0, 0, 0.5f);
560             GameObject.Find ("NH").GetComponent<SpriteRenderer> ().color = new
                   Color (1, 1, 0, 0.5f);
561             GameObject.Find ("Cercle").GetComponent<SpriteRenderer> ().color =
                   new Color (0, 1, 0, 0.5f);
562             if (currentState == "Move") {
563                 GameObject.Find ("M").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 0.5f);
564                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneM").transform.position;
565             } else if (currentState == "Crop") {
566                 GameObject.Find ("C").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 0.5f);
567                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneC").transform.position;
568             } else if (currentState == "Analisis") {
569                 GameObject.Find ("A").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 0.5f);
570                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneA").transform.position;
571             } else if (currentState == "Force") {
572                 GameObject.Find ("F").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 0.5f);
573                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneF").transform.position;
574             } else if (currentState == "Touch") {
575                 GameObject.Find ("T").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 0.5f);
576                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneT").transform.position;
577             } else if (currentState == "Paint") {
578                 GameObject.Find ("P").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 0.5f);
579                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneP").transform.position;
580             } else if (currentState == "AVib") {
581                 GameObject.Find ("AV").GetComponent<SpriteRenderer> ().color =
                       new Color (0, 1f, 0, 0.5f);
582                 GameObject.Find ("Cercle").transform.position =
```

```
                    GameObject.Find ("PlaneAV").transform.position;
583             } else if (currentState == "Instruccions") {
584                 GameObject.Find ("Inst").GetComponent<SpriteRenderer> ().color
                      = new Color (0, 1f, 0, 0.5f);
585                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneInst").transform.position;
586             } else if (currentState == "Sp") {
587                 GameObject.Find ("Sp").GetComponent<SpriteRenderer> ().color =
                      new Color (0, 1f, 0, 0.5f);
588                 GameObject.Find ("Cercle").transform.position =
                      GameObject.Find ("PlaneSp").transform.position;
589             }
590
591         }
592     }
593     public void Tancaselec (){
594         if (canvasseleccio.activeSelf) {
595             canvasseleccio.SetActive (false);
596         }
597         else{
598             canvasseleccio.SetActive (true);
599         }
600     }
601     public void Tancainfo (){
602         if (canvasinfo.activeSelf) {
603             canvasinfo.SetActive (false);
604         }
605         else if (canvasinfo2.activeSelf) {
606             canvasinfo2.SetActive (false);
607         }
608         else{
609             canvasinfo.SetActive (true);
610         }
611     }
612     public void next(){
613         if (canvasinfo.activeSelf) {
614             canvasinfo.SetActive (false);
615             canvasinfo2.SetActive (true);
616         }
617         else if (canvasinfo2.activeSelf) {
618             canvasinfo2.SetActive (false);
619             canvasinfo.SetActive (true);
620         }
621     }
622
623     void Entrardespresdetancarmenu(){
624         if (pastState=="Move") {
625             EnterMoveState();
626         }else if (pastState=="Crop"){
627             EnterCropState();
628         }
629         else if (pastState=="Analisis"){
630             EnterAnalisisState();
631         }
632         else if (pastState=="Force"){
633             EnterForceState();
```

```
634         }
635         else if (pastState=="Touch"){
636             EnterTouchState();
637         }
638         else if (pastState=="Paint"){
639             EnterPaintState();
640         }
641         else if (pastState=="AVib"){
642             EnterAVibState();
643         }
644     }
645     void canviestatdespresanalisis(){
646         if (currentState == "Analisis") {
647 
648             for (int i = 0; i < 2; i++)
649             {
650                 vr.transferFunction.ranges[i].hide = false;
651             }
652             for (int i=2; i< 6; i++)
653             {
654                 vr.transferFunction.ranges[i].hide = true;
655             }
656             vr.transferFunction.UpdateTexture();
657         }
658     }
659 
660 
661 
662 
663 }
664 
665 
```

Script Selec

F:\VRMedCode\Selec.cs 1

```
1  using UnityEngine;
2  using System.Collections;
3  using System.IO;
4  using UnityEngine.UI;
5  using System.Collections.Generic;
6  using UnityEditor;
7
8  public class Selec : MonoBehaviour {
9      public GameObject volume;
10     public Dropdown dropdown;
11     public Dropdown dropdown2;
12     private string directoryPath;
13     private string[] mostra;
14     private string[] mostra2;
15     string path;
16     List<string> llist = new List<string>() { "Tria una opció de funció de
         transferència" };
17     List<string> llist2 = new List<string>() { "Tria una opció de imatge" };
18     // Use this for initialization
19     void Start () {
20     //private int k=0
21
22         Llista();
23         Llista2();
24     }
25
26     // Update is called once per frame
27     void Update () {
28
29     }
30     void Llista (){
31         directoryPath = Application.dataPath + "/StreamingAssets/Palettes/";
32         mostra= Directory.GetFiles (@directoryPath, "*.plt");
33
34         foreach(string c in mostra){
35             llist.Add(Path.GetFileName(c));
36         }
37         dropdown.AddOptions(llist);
38
39     }
40     void Llista2()
41     {
42         directoryPath = Application.dataPath + "/StreamingAssets/";
43         mostra2 = Directory.GetFiles(@directoryPath, "*.pvm");
44
45         foreach (string d in mostra2)
46         {
47             llist2.Add(Path.GetFileName(d));
48         }
49         dropdown2.AddOptions(llist2);
50
51     }
52     public void Clica(int index){
53         if (index != 0) {
54             string path = mostra [index-1];
55
```

```
56              volume.GetComponent<VolumeRenderer>
                  ().transferFunction.LoadTransferFunction (path);
57          }
58
59      }
60      public void Clica2(int index)
61      {
62          if (index != 0) {
63              string path = mostra2 [index-1];
64
65              volume.GetComponent<VolumeRenderer> ().volume.LoadVolume (path);
66
67          }
68      }
69      public void busca(){
70          path = EditorUtility.OpenFilePanel ("Selecciona", "", "plt");
71          if (path != null)
72          {
73              volume.GetComponent<VolumeRenderer>
                  ().transferFunction.LoadTransferFunction(path);
74          }
75      }
76      public void busca2(){
77          path = EditorUtility.OpenFilePanel ("Selecciona", "", "pvm");
78          if (path != null) {
79              volume.GetComponent<VolumeRenderer>().volume.LoadVolume(path);
80          }
81      }
82
83
84  }
85
```

Script VolumeRenderer

F:\VRMedCode\VolumeRenderer.cs 1

```
1  using UnityEngine;
2  using System.Collections;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6
7  using RangeIndexPoint = System.Collections.Generic.KeyValuePair<int,
     UnityEngine.Vector3>;
8  using Polygon = System.Collections.Generic.KeyValuePair<UnityEngine.Vector3[],
     UnityEngine.Vector3[]>;
9
10 /// <summary>
11 /// Combines a <see cref="Volume"/> and a <see cref="TransferFunction"/> with
     a <see cref="Transform"/> to
12 /// get a renderable 3D volume.
13 /// </summary>
14 public class VolumeRenderer : MonoBehaviour
15 {
16     Vector3 a= new Vector3();
17     [SerializeField]
18     private Volume _volume;
19     /// <summary>
20     /// Reference to the rendered <see cref="Volume"/>.
21     /// </summary>
22     public Volume volume
23     {
24         get { return _volume; }
25         set
26         {
27             if (value != volume)
28             {
29                 _volume = value;
30                 essInitialized = false;
31                 essUpdated = meshesUpdated = cropMatrixUpdated = false;
32             }
33         }
34     }
35
36     [SerializeField]
37     private TransferFunction _transferFunction;
38     /// <summary>
39     /// Reference to the used <see cref="TransferFunction"/>.
40     /// </summary>
41     public TransferFunction transferFunction
42     {
43         get { return _transferFunction; }
44         set
45         {
46             if (value != transferFunction)
47             {
48                 _transferFunction = value;
49                 essUpdated = false;
50             }
51         }
52     }
53
```

```
54      [SerializeField]
55      private int _blockSize = 64;
56
57      /** @cond false */
58      public const int minBlockSize = 8;
59      public const int maxBlockSize = 128;
60      /** @endcond */
61
62      /// <summary>
63      /// Current ESS block size (automatically clamped to the nearest power of
          2).
64      /// </summary>
65      public int blockSize
66      {
67          get { return _blockSize; }
68          set
69          {
70              int val = (int)Mathf.Pow(2.0f, Mathf.Ceil(Mathf.Log((float)value,
                2.0f)));
71              _blockSize = Math.Max(minBlockSize, val);
72          }
73      }
74
75      /** @cond false */
76      [HideInInspector, NonSerialized]
77      public bool meshesUpdated = false;
78      /** @endcond */
79
80      private Mesh _renderMesh, _fullMesh;
81      /// <summary>
82      /// Mesh used on the rendering step.
83      /// </summary>
84      /// <code>renderMesh.uv</code> and <code>renderMes.uv2</code> conatin
85      /// normalized [0..1] vertex positions.
86      public Mesh renderMesh
87      {
88          get
89          {
90              if (!meshesUpdated) UpdateMeshes();
91              return _renderMesh;
92          }
93      }
94
95      /// <summary>
96      /// Full volume mesh, used for drawing Gizmos.
97      /// </summary>
98      public Mesh fullMesh
99      {
100         get
101         {
102             if (!meshesUpdated) UpdateMeshes();
103             return _fullMesh;
104         }
105     }
106
107     private void UpdateMeshes()
```

```
108     {
109
110         if (_renderMesh == null) _renderMesh = new Mesh();
111         if (_fullMesh == null) _fullMesh = new Mesh();
112
113         _renderMesh.Clear();
114         _fullMesh.Clear();
115
116         //Generate cube mesh
117         Vector3[] fullVerts = new Vector3[8];
118         Vector3[] verts = new Vector3[8];
119         Vector2[] normalizedVertsXY = new Vector2[8];
120         Vector2[] normalizedVertsZ = new Vector2[8];
121         Vector3 half = new Vector3(0.5f, 0.5f, 0.5f);
122
123
124         for (int i = 0; i < verts.Length; ++i)
125         {
126             fullVerts[i] = cubeVertices[i] - half;
127             CropBoxTransform(cubeVertices[i], out verts[i]);
128             normalizedVertsXY[i] = new Vector2(verts[i].x, verts[i].y);
129             normalizedVertsZ[i] = new Vector2(verts[i].z, 1.0f);
130             verts[i] -= half;
131
132             if (volume)
133             {
134                 verts[i] = Vector3.Scale(verts[i], volume.size);
135                 verts[i] += volume.volumePosition;
136
137                 fullVerts[i] = Vector3.Scale(fullVerts[i], volume.size);
138                 fullVerts[i] += volume.volumePosition;
139             }
140         }
141
142         _renderMesh.vertices = verts;
143         _renderMesh.uv = normalizedVertsXY; // Use uv and uv2 as a way to pass
               normalized position
144         _renderMesh.uv2 = normalizedVertsZ;
145         _renderMesh.triangles = cubeIndexes;
146         _renderMesh.RecalculateNormals();
147         _renderMesh.RecalculateBounds();
148
149         _fullMesh.vertices = fullVerts;
150         _fullMesh.triangles = cubeIndexes;
151         _fullMesh.RecalculateNormals();
152         _fullMesh.RecalculateBounds();
153
154         meshesUpdated = true;
155     }
156
157     private Polygon intersectPlane;
158     /// <summary>
159     /// Calculates intersection between <paramref name="camera"/>'s neacClip
          plane
160     /// and <see cref="renderMesh"/>.
161     /// </summary>
```

```
162     /// Calculates both vertex positions and their corresponding normalized
          [0..1] positions,
163     /// both sorted in counter-clockwise order.
164     /// <param name="camera"></param>
165     /// <param name="epsilon">Offset from <paramref name="camera"/>'s nearClip
          plane to the intersecting plane.</param>
166     /// <returns>A pair of <code>Vector3[]</code>, Key contains vertex
          positions and Value contains their corresponding normalized positions.</
          returns>
167     public Polygon GetCameraIntersectPlane(Camera camera, float epsilon)
168     {
169         Vector3 obs = camera.transform.position;
170         Vector3 normal = camera.transform.forward;
171 
172         Vector3 planeCenter = obs + (normal * (/*camera.nearClipPlane +*/
              epsilon));
173 
174         Plane plane = new Plane(normal, planeCenter);
175 
176         return GetIntersectPolygon(plane);
177     }
178 
179     public Polygon GetIntersectPolygon(Plane plane)
180     {
181         Mesh cube = renderMesh;
182         Vector3[] vertexs = new Vector3[8];
183 
184         bool[] side = new bool[8];
185         bool intersects = false;
186 
187         for (int i = 0; i < 8; i++)
188         {
189             vertexs[i] = transform.TransformPoint(cube.vertices[i]);
190             side[i] = plane.GetSide(vertexs[i]);
191             if (i > 0 && side[i] != side[i - 1]) intersects = true;
192         }
193 
194         if (!intersects)
195             return new Polygon(null, null);
196 
197         List<Vector3> points = new List<Vector3>();
198         foreach (IntPair e in cubeEdges)
199         {
200             int f = e.first, s = e.second;
201             if (side[f] != side[s])
202             {
203                 Ray r = new Ray(vertexs[f], vertexs[s] - vertexs[f]);
204                 points.Add(plane.Raycast(r));
205             }
206         }
207 
208         // We need to sort the points
209         Vector3 centroid = Vector3.zero;
210         foreach (Vector3 v in points) centroid += v;
211         centroid /= points.Count;
212 
```

```
            true;
260         if (bounds.Contains(transform.InverseTransformPoint(p55))) return
              true;
261
262         return false;
263     }
264
265     public Color SampleVolumeColor(Vector3 p, bool localCoords = false)
266     {
267
268         float v = SampleVolume(p, localCoords);
269         if (v < 0.0f) return Color.clear;
270         return transferFunction.texture.GetPixelBilinear(v, 0);
271
272     }
273
274     public float SampleVolume(Vector3 p, bool localCoords = false)
275     {
276         Vector3 local = p;
277         if (!localCoords)
278         {
279             local = transform.InverseTransformPoint(p);
280         }
281
282         if (!bounds.Contains(local)) return -1.0f;
283
284         local += volume.size / 2.0f;
285         p = local.DividedBy(volume.size).MultipliedBy(volume.resolution);
286
287        return volume.SampleVolume(p);
288     }
289     public Vector3 SampleGradient(Vector3 p) {
290         p = transform.InverseTransformPoint(p);
291         p += volume.size / 2.0f;
292         p = p.DividedBy(volume.size).MultipliedBy(volume.resolution);
293         return transform.TransformVector( volume.SampleGradientVolume(p));
294
295     }
296     public Vector3 SampleGradient2(Vector3 p) {
297         p = transform.InverseTransformPoint(p);
298         p += volume.size / 2.0f;
299         p = p.DividedBy(volume.size).MultipliedBy(volume.resolution);
300         p = volume.SampleGradientVolume (p);
301         p [0] = (p [0] - 0.5f) * 2.0f;
302         p [1] = (p [1] - 0.5f) * 2.0f;
303         p [2] = (p [2] - 0.5f) * 2.0f;
304         return transform.TransformVector( p);
305
306     }
307     public Vector3 SampleGradient3(Vector3 p) {
308         p = transform.InverseTransformPoint(p);
309         p += volume.size / 2.0f;
310         p = p.DividedBy(volume.size).MultipliedBy(volume.resolution);
311         return transform.TransformVector( volume.SampleGradientVolumeMatrix
              (p));
312
```

```
313     }
314     public Vector3 Vectorcercle ()
315     {
316         Vector3 p2;
317         p2.x = 0;
318         p2.y = 0;
319         p2.z = 0;
320         return transform.TransformPoint (p2);
321     }
322     public Vector3 SampleGradient_Malo(Vector3 p)
323     {
324 
325         p = transform.InverseTransformPoint(p);
326         p += volume.size / 2.0f; ;
327         p = p.DividedBy(volume.size).MultipliedBy(volume.resolution);
328         Vector3 voxelSize = volume.resolution.DividedBy(volume.size);
329 
330         float intervalInferior = 0f;
331         float intervalSuperior = 1f;
332 
333 
334 
335         float mx = volume.SampleVolume(p - Vector3.right * voxelSize.x);
336         mx = mx > intervalInferior ? mx : 0;
337         mx = mx < intervalSuperior ? mx : 0;
338         float Mx = volume.SampleVolume(p + Vector3.right * voxelSize.x);
339         Mx = Mx > intervalInferior ? Mx : 0;
340         Mx = Mx < intervalSuperior ? Mx : 0;
341         //print("mx:" + mx.ToString()+ "  Mx:" + Mx.ToString());
342 
343         float my = volume.SampleVolume(p - Vector3.up * voxelSize.y);
344         my = my > intervalInferior ? my : 0;
345         my = my < intervalSuperior ? my : 0;
346         float My = volume.SampleVolume(p + Vector3.up * voxelSize.y);
347         My = My > intervalInferior ? My : 0;
348         My = My < intervalSuperior ? My : 0;
349 
350         float mz = volume.SampleVolume(p - Vector3.forward * voxelSize.z);
351         mz = mz > intervalInferior ? mz : 0;
352         mz = mz < intervalSuperior ? mz : 0;
353         float Mz = volume.SampleVolume(p + Vector3.forward * voxelSize.z);
354 
355         Mz = Mz > intervalInferior ? Mz : 0;
356         Mz = Mz < intervalSuperior ? Mz : 0;
357 
358 
359         Vector3 gradient = new Vector3((mx - Mx), (my - My), (mz - Mz));
360         return -transform.TransformVector(gradient);
361 
362     }
363 
364 
365 
366     public bool Pint (Vector3 p){
367         Vector3 local = p;
368         local = transform.InverseTransformPoint(p);
```

```
369         if (!bounds.Contains (local)) {
370             return false;
371         }
372         local += volume.size / 2.0f;
373         p = local.DividedBy(volume.size).MultipliedBy(volume.resolution);
374         if (volume.SampleVolume(p) < .3f) {
375             return false;
376         }
377         else{
378             return true;
379         }
380     }
381     public class RangePointComparer : EqualityComparer<RangeIndexPoint>
382     {
383         public override bool Equals(RangeIndexPoint x, RangeIndexPoint y)
384         {
385             return x.Key == y.Key;
386         }
387 
388         public override int GetHashCode(RangeIndexPoint obj)
389         {
390             return obj.Key.GetHashCode();
391         }
392     }
393 
394     public RangeIndexPoint[] GetRanges(Ray r, bool localCoords = false)
395     {
396         var result = new List<RangeIndexPoint>();
397 
398         if (!localCoords)
399         {
400             r = transform.InverseTransformRay(r);
401         }
402 
403         Vector3 vs = volume.size.DividedBy(volume.resolution);
404         float delta = vs.Min();
405 
406         Vector3 p;
407         float distance;
408         if (bounds.Contains(r.origin))
409         {
410             distance = 0.0f;
411             p = r.origin;
412         }
413         else if (!bounds.IntersectRay(r, out distance))
414         {
415             return new RangeIndexPoint[0];
416         }
417         else
418         {
419             p = r.GetPoint(distance + delta);
420         }
421 
422         int lastRange = -1;
423         while (bounds.Contains(p))
424         {
```

```
425             float v = SampleVolume(p, true);
426
427             float maxAlpha = -1.0f;
428             int currentRange = -1;
429
430             for (int i = 0; i < transferFunction.ranges.Count; ++i)
431             {
432                 TFRange range = transferFunction.ranges[i];
433
434                 //if (range.curve.keys[0].time > v || range.curve.keys[3].time
                       < v) continue;
435
436                 float alpha = 0;//range.curve.Evaluate(v);
437                 if (alpha > maxAlpha)
438                 {
439                     currentRange = i;
440                     maxAlpha = alpha;
441                 }
442             }
443
444             if (currentRange != lastRange)
445             {
446                 if (currentRange != -1) result.Add(new RangeIndexPoint
                      (currentRange,p));
447                 lastRange = currentRange;
448             }
449
450             distance += delta;
451             p += r.direction * delta;
452         }
453
454         var finalResult = new List<RangeIndexPoint>();
455         for (int i = 0; i < result.Count; ++i)
456         {
457             if (i == 0 || i == result.Count - 1)
458             {
459                 finalResult.Add(result[i]);
460                 continue;
461             }
462
463             var prev = result[i - 1];
464             var curr = result[i];
465             var next = result[i + 1];
466
467             if (curr.Key != prev.Key
468                 && (Vector3.Distance(prev.Value, curr.Value) > delta * 10.0f)
469                 && (Vector3.Distance(curr.Value, next.Value) > delta * 10.0f))
470             {
471                 finalResult.Add(curr);
472             }
473         }
474
475         return finalResult.ToArray(); //.Distinct(new RangePointComparer
              ()).ToArray();
476     }
477
```

```
478     private Vector3 blocks;
479     private byte[,,] blocksMin;
480     private byte[,,] blocksMax;
481
482     /** @cond false */
483     [HideInInspector, NonSerialized]
484     public bool essInitialized = false;
485     [HideInInspector, NonSerialized]
486     public bool essUpdated = false;
487     /** @endcond */
488
489     private Mesh[] _essMeshes;
490     /// <summary>
491     /// Array of meshes used in the rendering process if Empty Space Skipping
          is enabled.
492     /// </summary>
493     /// Only occupied cubes are included and individual cube meshes are
          combined into as
494     /// few meshes as possible.
495     public Mesh[] essMeshes
496     {
497         get
498         {
499             if (!essInitialized) InitESS();
500             if (!essUpdated) UpdateESS();
501             return _essMeshes;
502         }
503     }
504
505     private void InitESS()
506     {
507         if (!Application.isPlaying) { Debug.LogWarning("Can't init ESS in
              Editor."); return; }
508         Vector3 b = volume.resolution/blockSize;
509
510         int blocksX = Mathf.CeilToInt(b.x);
511         int blocksY = Mathf.CeilToInt(b.y);
512         int blocksZ = Mathf.CeilToInt(b.z);
513
514         blocks = new Vector3(blocksX, blocksY, blocksZ);
515
516         blocksMin = new byte[blocksZ, blocksY, blocksX];
517         blocksMax = new byte[blocksZ, blocksY, blocksX];
518
519         Color32[] texels = volume.volumeTexture.GetPixels32();
520
521         for (int i = 0; i < blocks.z; i++)
522         {
523             for (int j = 0; j < blocks.y; j++)
524             {
525                 for (int k = 0; k < blocks.x; k++)
526                 {
527                     BlockMinMax(texels, i, j, k);
528                 }
529             }
530         }
```

```
531
532         essInitialized = true;
533         essUpdated = false;
534     }
535
536     private void BlockMinMax(Color32[] texels, int i, int j, int k)
537     {
538         Vector3 res = volume.resolution;
539         long resx = (long)res.x, resy = (long)res.y, resz = (long)res.z;
540         byte min = Byte.MaxValue;
541         byte max = Byte.MinValue;
542
543         for (int ii = 0; ii < blockSize; ii++)
544         {
545             for (int jj = 0; jj < blockSize; jj++)
546             {
547                 for (int kk = 0; kk < blockSize; kk++)
548                 {
549                     int x = k * blockSize + kk;
550                     int y = j * blockSize + jj;
551                     int z = i * blockSize + ii;
552
553                     if (x < resx && y < resy && z < resz)
554                     {
555                         long texelPos = z * resx * resy + y * resx + x;
556                         byte value = texels[texelPos].a;
557                         if (value < min) min = value;
558                         if (value > max) max = value;
559                     }
560                 }
561             }
562         }
563         blocksMin[i, j, k] = min;
564         blocksMax[i, j, k] = max;
565     }
566
567     private const int maxCubesPerMesh = 8191;
568     private void UpdateESS()
569     {
570         if (!essInitialized) return;
571
572         List<Mesh> meshes = new List<Mesh>();
573
574         Vector3 inverseBlocks = blocks.Inverted();
575         Vector3 half = new Vector3(0.5f, 0.5f, 0.5f);
576
577         for (int i = 0; i < blocks.z; i++)
578         {
579             for (int j = 0; j < blocks.y; j++)
580             {
581                 for (int k = 0; k < blocks.x; k++)
582                 {
583                     bool isOccupied = OccupiedBlock(i,j,k);
584                     if (!isOccupied) continue;
585
586                     Vector3 offset = new Vector3(k, j, i);
```

```
587                     Vector3 min = Vector3.Scale(offset, inverseBlocks);
588                     Vector3 max = Vector3.Scale(offset + Vector3.one,
                    inverseBlocks);
589
590                     if (cropXmin > max.x || cropXmax < min.x ||
591                         cropYmin > max.y || cropYmax < min.y ||
592                         cropZmin > max.z || cropZmax < min.z) continue;
593
594                     bool visible =
595                        (min.x <= cropXmin && cropXmin <= max.x ||
596                         min.y <= cropYmin && cropYmin <= max.y ||
597                         min.z <= cropZmin && cropZmin <= max.z ||
598
599                         min.x <= cropXmax && cropXmax <= max.x ||
600                         min.y <= cropYmax && cropYmax <= max.y ||
601                         min.z <= cropZmax && cropZmax <= max.z);
602
603                     bool covered =
604                         (k > 0 && k < blocks.x - 1 &&
605                         j > 0 && j < blocks.y - 1 &&
606                         i > 0 && i < blocks.z - 1 &&
607                         OccupiedBlock(i, j, k + 1) &&
608                         OccupiedBlock(i, j, k - 1) &&
609                         OccupiedBlock(i, j + 1, k) &&
610                         OccupiedBlock(i, j - 1, k) &&
611                         OccupiedBlock(i + 1, j, k) &&
612                         OccupiedBlock(i - 1, j, k));
613
614                     if (covered && !visible) continue;
615
616                     Mesh m = new Mesh();
617
618                     //Generate cube mesh
619                     Vector3[] verts = new Vector3[8];
620                     Vector2[] normalizedVertsXY = new Vector2[8];
621                     Vector2[] normalizedVertsZ = new Vector2[8];
622
623                     for (int vi = 0; vi < verts.Length; ++vi)
624                     {
625                         verts[vi] = Vector3.Scale((cubeVertices[vi] +
                    offset),inverseBlocks);
626                         verts[vi] = new Vector3(Mathf.Clamp(verts[vi].x,
                    cropXmin, cropXmax),
627                             Mathf.Clamp(verts[vi].y, cropYmin, cropYmax),
628                             Mathf.Clamp(verts[vi].z, cropZmin, cropZmax));
629
630                         normalizedVertsXY[vi] = new Vector2(verts[vi].x,verts
                    [vi].y);
631                         normalizedVertsZ[vi] = new Vector2(verts[vi].z, 1.0f);
632                         verts[vi] -= half;
633
634                         if (volume)
635                         {
636                             verts[vi] = Vector3.Scale(verts[vi], volume.size);
637                             verts[vi] += volume.volumePosition;
638                         }
```

```
639                     }
640
641                     m.vertices = verts;
642                     m.uv = normalizedVertsXY; // Use uv and uv2 as a way to
                        pass normalized position
643                     m.uv2 = normalizedVertsZ;
644                     m.triangles = cubeIndexes;
645
646                     meshes.Add(m);
647                 }
648             }
649         }
650
651         int nMeshes = (meshes.Count + maxCubesPerMesh - 1) / maxCubesPerMesh;
652
653         _essMeshes = new Mesh[nMeshes];
654
655         for (int i = 0; i < nMeshes; ++i)
656         {
657             int n = Math.Min(maxCubesPerMesh, meshes.Count - maxCubesPerMesh *
                    i);
658             CombineInstance[] combine = new CombineInstance[n];
659
660             for (int j = 0; j < n; ++j)
661             {
662                 combine[j].mesh = meshes[maxCubesPerMesh*i + j];
663                 combine[j].transform = Matrix4x4.identity;
664             }
665
666             _essMeshes[i] = new Mesh();
667             _essMeshes[i].CombineMeshes(combine);
668         }
669
670         foreach (Mesh m in meshes) Destroy(m);
671
672         essUpdated = true;
673     }
674
675     private bool OccupiedBlock(int i, int j, int k)
676     {
677         return transferFunction.RangeContainsData(blocksMin[i, j, k],
              blocksMax[i, j, k]);
678     }
679
680     //Draws fullMesh and renderMesh in editor
681     /** @cond false */
682     public void OnDrawGizmos()
683     {
684
685         Color c = Gizmos.color;
686
687         Gizmos.color = new Color(1.0f, 0.92f, 0.016f, 0.05f);
688         Gizmos.DrawWireMesh(fullMesh, transform.position, transform.rotation,
              transform.localScale);
689
690         Gizmos.color = Color.yellow;
```

```
691         Gizmos.DrawWireMesh
              (renderMesh,transform.position,transform.rotation,transform.localSca
              le);
692
693         Gizmos.color = c;
694     }
695     /** @endcond */
696
697     /*
698      * Crop Range
699      */
700     [SerializeField]
701     private float _cropXmin = 0.0f, _cropYmin = 0.0f, _cropZmin = 0.0f;
702     [SerializeField]
703     private float _cropXmax = 1.0f, _cropYmax = 1.0f, _cropZmax = 1.0f;
704
705
706     /// Crop range's lower bound of the X axis
707     public float cropXmin { get { return _cropXmin; } set { SetCropFloat(ref
          _cropXmin, value); } }
708     /// Crop range's lower bound of the Y axis
709     public float cropYmin { get { return _cropYmin; } set { SetCropFloat(ref
          _cropYmin, value); } }
710     /// Crop range's lower bound of the Z axis
711     public float cropZmin { get { return _cropZmin; } set { SetCropFloat(ref
          _cropZmin, value); } }
712
713     /// Crop range's upper bound of the X axis
714     public float cropXmax { get { return _cropXmax; } set { SetCropFloat(ref
          _cropXmax, value); } }
715     /// Crop range's upper bound of the Y axis
716     public float cropYmax { get { return _cropYmax; } set { SetCropFloat(ref
          _cropYmax, value); } }
717     /// Crop range's upper bound of the Z axis
718     public float cropZmax { get { return _cropZmax; } set { SetCropFloat(ref
          _cropZmax, value); } }
719
720     private void SetCropFloat(ref float f, float value)
721     {
722         float v = Mathf.Clamp01(value);
723
724         f = v;
725
726         essUpdated = meshesUpdated = cropMatrixUpdated = false;
727
728     }
729
730     private void CropBoxTransform(Vector3 vertex, out Vector3 v)
731     {
732         float x = vertex.x > 0.5f ? vertex.x * cropXmax : cropXmin;
733         float y = vertex.y > 0.5f ? vertex.y * cropYmax : cropYmin;
734         float z = vertex.z > 0.5f ? vertex.z * cropZmax : cropZmin;
735
736         v = new Vector3(x, y, z);
737     }
738
```

```
739     private bool cropMatrixUpdated = false;
740     private Matrix4x4 _cropMatrix;
741     /// <summary>
742     /// Crop range values in a <see cref="Matrix4x4"/>, ready to use in
          Raytrace shader.
743     /// </summary>
744     /// First colum contains lower bounds and second column contains upper
          bounds (all in range [0..1]).
745     public Matrix4x4 cropMatrix
746     {
747         get
748         {
749             if (!cropMatrixUpdated) UpdateCropMatrix();
750             return _cropMatrix;
751         }
752     }
753
754     private void UpdateCropMatrix()
755     {
756         if (!volume.isValid)
757         {
758             _cropMatrix.SetColumn(0, new Vector4(cropXmin, cropYmin, cropZmin,
                0));
759             _cropMatrix.SetColumn(1, new Vector4(cropXmax, cropYmax, cropZmax,
                0));
760         }
761         else
762         {
763             Vector3 o = volume.texIncrement;
764             _cropMatrix.SetColumn(0, new Vector4(cropXmin + o.x, cropYmin +
                o.y, cropZmin + o.z, 0));
765             _cropMatrix.SetColumn(1, new Vector4(cropXmax - o.x, cropYmax -
                o.y, cropZmax - o.z, 0));
766
767         }
768         cropMatrixUpdated = true;
769     }
770
771     /// <summary>
772     /// Bounds of the rendered volume (in local coordinate system).
773     /// </summary>
774     public Bounds bounds
775     {
776         get
777         {
778             if (!meshesUpdated) UpdateMeshes();
779             return renderMesh.bounds;
780         }
781     }
782
783     /*
784      * Cube mesh data
785      */
786     #region MeshData
787
788     private readonly Vector3[] cubeVertices = new[] {
```

```
789         new Vector3(0f,0f,1f),
790         new Vector3(1f,0f,1f),
791         new Vector3(1f,1f,1f),
792         new Vector3(0f,1f,1f),
793
794         new Vector3(0f,0f,0f),
795         new Vector3(1f,0f,0f),
796         new Vector3(1f,1f,0f),
797         new Vector3(0f,1f,0f)
798     };
799
800     private readonly int[] cubeIndexes = new[]
801     {
802         0, 1, 2,
803         2, 3, 0,
804         // top
805         1, 5, 6,
806         6, 2, 1,
807         // back
808         7, 6, 5,
809         5, 4, 7,
810         // bottom
811         4, 0, 3,
812         3, 7, 4,
813         // left
814         4, 5, 1,
815         1, 0, 4,
816         // right
817         3, 2, 6,
818         6, 7, 3
819     };
820
821
822     private class IntPair { public int first, second; }
823     /// <summary>
824     /// Definition of edges used in <see cref="GetCameraIntersectPlane
        (Camera)"/>.
825     /// </summary>
826     private readonly IntPair[] cubeEdges = new IntPair[]
827     {
828         new IntPair() { first = 0, second = 1 },
829         new IntPair() { first = 0, second = 4 },
830         new IntPair() { first = 0, second = 3 },
831
832         new IntPair() { first = 6, second = 2 },
833         new IntPair() { first = 6, second = 5 },
834         new IntPair() { first = 6, second = 7 },
835
836         new IntPair() { first = 1, second = 2 },
837         new IntPair() { first = 2, second = 3 },
838         new IntPair() { first = 3, second = 7 },
839         new IntPair() { first = 7, second = 4 },
840         new IntPair() { first = 4, second = 5 },
841         new IntPair() { first = 5, second = 1 }
842     };
843
```

```
844         #endregion
845     }
```

Script Volume

F:\VRMedCode\Volume.cs 1

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Runtime.InteropServices;
4  using System;
5  #if UNITY_EDITOR
6  using UnityEditor;
7  #endif
8
9
10
11
12 /// <summary>
13 /// Holds volumetric model data.
14 /// </summary>
15 [Serializable]
16 public class Volume : ScriptableObject
17 {
18     // (1024 * 1024 * 1024) / 4 = 1GB / 4 bytes per pixel
19     private const ulong MAX_DATA_SIZE = (1024 * 1024 * 1024) / 16;
20     private const int maxPreviewSize = 128;
21     public int[] res = new int[3];
22     [SerializeField]
23     private Texture3D _volumeTexture;
24
25     //Stored data
26     /// <summary>
27     /// 3D texture holding the property Alpha8 values.
28     /// </summary>
29     public Texture3D volumeTexture { get { return _volumeTexture; } }
30
31     private Color32[] _pixels = null;
32     public Color32[] pixels
33     {
34         get
35         {
36             if (_pixels == null || _pixels.Length == 0)
37             {
38                 _pixels = volumeTexture.GetPixels32();
39             }
40             return _pixels;
41         }
42     }
43
44     private Color32[] _pixelsgrad = null;
45     public Color32[] pixelsgrad
46     {
47         get
48         {
49             if (_pixelsgrad == null || _pixelsgrad.Length == 0)
50             {
51                 _pixelsgrad = gradientTexture.GetPixels32();
52             }
53             return _pixelsgrad;
54         }
55     }
56
```

```
57
58      [SerializeField]
59      private Texture3D _gradientTexture;
60      /// <summary>
61      /// 3D texture holding gradients calculated from <see cref="volumeTexture"/>.
62      /// </summary>
63      public Texture3D gradientTexture { get { return _gradientTexture; } }
64
65      /// <summary>
66      /// Volume's additional traslation.
67      /// </summary>
68      public Vector3 volumePosition;
69
70      /// <summary>
71      /// Size of each individual voxel (in mm).
72      /// </summary>
73      public Vector3 voxelSize;
74
75      /// <summary>
76      /// Maximum gradient maginute in <see cref="gradientTexture"/>.
77      /// </summary>
78      /// Needed to de-normalize magintude in the texture (alpha channel).
79      public float maxGradient;
80
81      /// <summary>
82      /// Volume's 3D texture resolution.
83      /// </summary>
84      public Vector3 resolution
85      {
86          get
87          {
88              if (!isValid) return Vector3.zero;
89              return new Vector3(volumeTexture.width, volumeTexture.height, volumeTexture.depth);
90          }
91      }
92      /// <summary>
93      /// Inverse of <see cref="resolution"/>.
94      /// </summary>
95      public Vector3 texIncrement
96      {
97          get
98          {
99              if (!isValid) return Vector3.zero;
100             return new Vector3(1.0f / volumeTexture.width, 1.0f / volumeTexture.height, 1.0f / volumeTexture.depth);
101         }
102     }
103
104     public float minTexIncrement
105     {
106         get { return Mathf.Min(texIncrement.x, texIncrement.y, texIncrement.z); }
107     }
108
```

```
109     /// <summary>
110     /// Size of the volume in game units (without scaling).
111     /// </summary>
112     public Vector3 size
113     {
114         get
115         {
116             if (!isValid) return Vector3.zero;
117             return Vector3.Scale(resolution, voxelSize) / 1000.0f;
118         }
119     }
120
121     /// <summary>
122     /// True if the volume is properly loaded, false otherwise.
123     /// </summary>
124     public bool isValid
125     {
126         get
127         {
128             return volumeTexture != null;
129         }
130     }
131
132     [SerializeField]
133     private Texture2D _previewTexture;
134
135     /// <summary>
136     /// Small 2D texture that can be used as a preview of the volume.
137     /// </summary>
138     public Texture2D previewTexture
139     {
140         get
141         {
142             if (!isValid) return null;
143             return _previewTexture;
144         }
145     }
146
147     public Vector3 SampleGradientVolume(Vector3 p)
148     {
149         int px = (int)p.x; int py = (int)p.y; int pz = (int)p.z;
150
151         long index = pz * volumeTexture.width * volumeTexture.height + py *
              volumeTexture.width + px;
152         if (index >= pixelsgrad.Length) return Vector3.zero;
153         if (index < 0) return Vector3.zero;
154
155         return new Vector3 (pixelsgrad[index].r, pixelsgrad[index].g,
              pixelsgrad[index].b) /255f;
156
157     }
158
159     public float SampleVolume(Vector3 p)
160     {
161         int px = (int)p.x; int py = (int)p.y; int pz = (int)p.z;
162         long index = pz * volumeTexture.width * volumeTexture.height + py *
```

```
            volumeTexture.width + px;
163         Color32[] pix = pixels;
164         if (index >= pix.Length) return 0;
165         if (index < 0) return 0;
166         return pix[index].a / 255.0f;
167     }
168
169
170     /// <summary>
171     /// Sample the volume texture given a <see cref="Vector3"/> in the range
        [0..<see cref="resolution"/>].
172     /// </summary>
173     /// <param name="p">Must be in the range [0..<see cref="resolution"/></
        param>
174     /// <returns>Value of the texture at point <paramref name="p"/>.  </
        returns>
175     public float SampleVolumeTriliniar(Vector3 p)
176     {
177         if (p.x < 0 || p.x >= volumeTexture.width ||
178             p.y < 0 || p.y >= volumeTexture.width ||
179             p.z < 0 || p.z >= volumeTexture.width) return 0;
180
181         Vector3 voxelOrigin = new Vector3(Mathf.Floor(p.x), Mathf.Floor(p.y),
              Mathf.Floor(p.z));
182
183         float incrx, incry, incrz;
184         incrx = p.x - voxelOrigin.x;
185         incry = p.y - voxelOrigin.y;
186         incrz = p.z - voxelOrigin.z;
187
188         incrx /= voxelSize.x;
189         incry /= voxelSize.y;
190         incrz /= voxelSize.z;
191
192         int x = 0, y = 0, z = 0;
193
194         if (incrx > 0)
195         {
196             if (voxelOrigin.x < resolution.x - 1) x = 1;
197         }
198         else
199         {
200             if (voxelOrigin.x > 0) x = -1;
201             incrx *= -1;
202         }
203
204         if (incry > 0)
205         {
206             if (voxelOrigin.y < resolution.y - 1) y = 1;
207         }
208         else
209         {
210             if (voxelOrigin.y > 0) y = -1;
211             incry *= -1;
212         }
213
```

```
214         if (incrz > 0)
215         {
216             if (voxelOrigin.z < resolution.z - 1) z = 1;
217         }
218         else
219         {
220             if (voxelOrigin.z > 0) z = -1;
221             incrz *= -1;
222         }
223         float[] values = new float[8];
224 
225         values[0] = SampleVolume(voxelOrigin + new Vector3(0, 0, 0)) / 255.0f;
226         values[1] = SampleVolume(voxelOrigin + new Vector3(x, 0, 0)) / 255.0f;
227         values[4] = SampleVolume(voxelOrigin + new Vector3(x, y, 0)) / 255.0f;
228         values[7] = SampleVolume(voxelOrigin + new Vector3(x, y, z)) / 255.0f;
229         values[5] = SampleVolume(voxelOrigin + new Vector3(x, 0, z)) / 255.0f;
230         values[2] = SampleVolume(voxelOrigin + new Vector3(0, y, 0)) / 255.0f;
231         values[6] = SampleVolume(voxelOrigin + new Vector3(0, y, z)) / 255.0f;
232         values[3] = SampleVolume(voxelOrigin + new Vector3(0, 0, z)) / 255.0f;
233 
234 
235         float _incrx = 1.0f - incrx;
236         float _incry = 1.0f - incry;
237         float _incrz = 1.0f - incrz;
238 
239         return
240             (_incrx * _incry * _incrz * values[0] +
241               incrx * _incry * _incrz * values[1] +
242              _incrx *  incry * _incrz * values[2] +
243              _incrx * _incry *  incrz * values[3] +
244               incrx *  incry * _incrz * values[4] +
245               incrx * _incry *  incrz * values[5] +
246              _incrx *  incry *  incrz * values[6] +
247               incrx *  incry *  incrz * values[7]);
248     }
249 
250 
251 
252     //DLL imports
253     [DllImport("libvrmed")]
254     private static extern bool LoadFile(string path,
255         ref float posx, ref float posy, ref float posz,
256         ref uint resx, ref uint resy, ref uint resz,
257         ref float sizex, ref float sizey, ref float sizez,
258         ref IntPtr ptrResult);
259 
260     [DllImport("libvrmed")]
261     private static extern void FreeMem(IntPtr ptr);
262 
263     /// <summary>
264     /// Loads volume data from a file in PVM format.
265     /// </summary>
266     /// <param name="filePath">Path of the file to be loaded.</param>
267     /// <returns>True if file loaded properly, false otherwise.</returns>
268     public bool LoadVolume(string filePath)
269     {
```

```
270         Debug.Log(filePath);
271
272         float posx, posy, posz;
273         uint resx, resy, resz;
274         float sizex, sizey, sizez;
275         IntPtr dataPtr;
276
277         posx = posy = posz = 0f;
278         resx = resy = resz = 0;
279         sizex = sizey = sizez = 0f;
280         dataPtr = IntPtr.Zero;
281
282         bool success = LoadFile(filePath,
283             ref posx, ref posy, ref posz,
284             ref resx, ref resy, ref resz,
285             ref sizex, ref sizey, ref sizez,
286             ref dataPtr);
287
288         if (!success)
289         {
290             Debug.LogError("Cannot load volume.");
291             _volumeTexture = null;
292             return false;
293         }
294
295         while (resx * resy * resz > MAX_DATA_SIZE)
296         {
297             resz /= 2;
298         }
299
300
301         volumePosition = new Vector3(posx, posy, posz);
302
303         uint dataSize = resx * resy * resz;
304
305         byte[] rawData = new byte[dataSize];
306
307         Marshal.Copy(dataPtr, rawData, 0, (int)dataSize);
308         FreeMem(dataPtr);
309
310         Color32[] pixels32 = new Color32[dataSize];
311         res[0] = Convert.ToInt32(resx);
312         res[1] = Convert.ToInt32(resy);
313         res[2] = Convert.ToInt32(resz);
314
315         for (int i = 0; i < resz; ++i)
316         {
317             for (int j = 0; j < resy; ++j)
318             {
319                 for (int k = 0; k < resx; ++k)
320                 {
321                     long pixelPos = i * resx * resy + j * resx + k;
322                     long texPos = i * resx * resy + j * resx + ((resx - 1) -
                        k);
323                     pixels32[pixelPos] = new Color32(0, 0, 0, rawData
                        [texPos]);
```

```
324                     //pixels32[pixelPos] = new Color32(0, 0, 0, (byte)  (i >
                        100 ? 255 : 0));
325                 }
326             }
327         }
328
329         _volumeTexture = new Texture3D((int)resx, (int)resy, (int)resz,
              TextureFormat.Alpha8, false)
330         {
331             wrapMode = TextureWrapMode.Clamp,
332             filterMode = FilterMode.Bilinear
333         };
334         volumeTexture.SetPixels32(pixels32);
335         volumeTexture.Apply();
336
337         Color[] gradients = new Color[dataSize];
338         maxGradient = 0.0f;
339
340         for (int i = 0; i < resz; ++i)
341         {
342             for (int j = 0; j < resy; ++j)
343             {
344                 for (int k = 0; k < resx; ++k)
345                 {
346                     int mx = (k > 0) ? (int)rawData[i * resx * resy + j * resx
                         + (resx - 1) - (k - 1)] : 0;
347                     int my = (j > 0) ? (int)rawData[i * resx * resy + (j - 1)
                        * resx + (resx - 1) - k] : 0;
348                     int mz = (i > 0) ? (int)rawData[(i - 1) * resx * resy + j
                        * resx + (resx - 1) - k] : 0;
349
350                     int Mx = (k < resx - 1) ? (int)rawData[i * resx * resy + j
                         * resx + (resx - 1) - (k + 1)] : 0;
351                     int My = (j < resy - 1) ? (int)rawData[i * resx * resy +
                        (j + 1) * resx + (resx - 1) - k] : 0;
352                     int Mz = (i < resz - 1) ? (int)rawData[(i + 1) * resx *
                        resy + j * resx + (resx - 1) - k] : 0;
353
354                     Vector3 gradient = new Vector3(mx - Mx, my - My, mz -
                        Mz) / 255;
355
356                     float magnitude = gradient.magnitude;
357                     maxGradient = Mathf.Max(magnitude, maxGradient);
358                 }
359             }
360         }
361
362         for (int i = 0; i < resz; ++i)
363         {
364             for (int j = 0; j < resy; ++j)
365             {
366                 for (int k = 0; k < resx; ++k)
367                 {
368                     long pixelPos = i * resx * resy + j * resx + k;
369
370                     int mx = (k > 0) ? (int)rawData[i * resx * resy + j * resx
```

```
                        + (resx - 1) - (k - 1)] : 0;
371                     int my = (j > 0) ? (int)rawData[i * resx * resy + (j - 1)
                        * resx + (resx - 1) - k] : 0;
372                     int mz = (i > 0) ? (int)rawData[(i - 1) * resx * resy + j
                        * resx + (resx - 1) - k] : 0;
373
374                     int Mx = (k < resx - 1) ? (int)rawData[i * resx * resy + j
                         * resx + (resx - 1) - (k + 1)] : 0;
375                     int My = (j < resy - 1) ? (int)rawData[i * resx * resy +
                        (j + 1) * resx + (resx - 1) - k] : 0;
376                     int Mz = (i < resz - 1) ? (int)rawData[(i + 1) * resx *
                        resy + j * resx + (resx - 1) - k] : 0;
377
378                     Vector3 gradient = new Vector3(mx - Mx, my - My, mz -
                        Mz) / 255;
379
380                     float magnitude = gradient.magnitude;
381                     gradient.Normalize();
382
383                     gradients[pixelPos] = new Color(gradient.x * 0.5f + 0.5f,
                        gradient.y * 0.5f + 0.5f, gradient.z * 0.5f + 0.5f,
                        magnitude / maxGradient);
384                 }
385             }
386         }
387
388         _gradientTexture = new Texture3D((int)resx, (int)resy, (int)resz,
              TextureFormat.RGBA32, false)
389         {
390             wrapMode = TextureWrapMode.Clamp,
391             filterMode = FilterMode.Point
392         };
393         gradientTexture.SetPixels(gradients);
394         gradientTexture.Apply();
395
396         voxelSize = new Vector3(sizex, sizey, sizez);
397
398         volumePosition = new Vector3(posx, posy, posz);
399
400         _pixels = null;
401
402         MakePreview();
403
404         return true;
405     }
406
407
408     #region
409     public Vector3 SampleGradientVolumeNew(Vector3 p)
410     {
411         int k = (int)p.x; int j= (int)p.y; int i = (int)p.z;
412         int resx, resy, resz;
413         resx = volumeTexture.width;
414         resy = volumeTexture.height;
415         resz = volumeTexture.depth;
416         Color32[] pix = pixels;
```

```
417         Vector3 p2;
418         p2.x = k-1;
419         p2.y = j;
420         p2.z = i;
421         float mx = (k > 0) ? SampleVolume(p2): 0 ;
422         p2.x = k;
423         p2.y = j-1;
424         p2.z = i;
425         float my = (j > 0) ?  SampleVolume(p2) : 0;
426         p2.x = k;
427         p2.y = j;
428         p2.z = i-1;
429         float mz = (i > 0) ?  SampleVolume(p2): 0;
430
431         p2.x = k+1;
432         p2.y = j;
433         p2.z = i;
434         float Mx = (k < resx - 1) ? SampleVolume(p2): 0;
435         p2.x = k;
436         p2.y = j+1;
437         p2.z = i;
438         float My = (j < resy - 1) ? SampleVolume(p2): 0;
439         p2.x = k;
440         p2.y = j;
441         p2.z = i+1;
442         float Mz = (i < resz - 1) ? SampleVolume(p2): 0;
443
444         Vector3 gradient = new Vector3((mx - Mx)/voxelSize.x, (my - My)/
              voxelSize.y, (mz - Mz)/voxelSize.z);
445         gradient.Normalize();
446         return gradient;
447     }
448     public Vector3 SampleGradientVolumeMean2(Vector3 p)
449     {
450         int k = (int)p.x; int j= (int)p.y; int i = (int)p.z;
451         int contador = 1;
452         Vector3 p2, a=Vector3.zero, a1=Vector3.zero, a2=Vector3.zero,
              a3=Vector3.zero, a4=Vector3.zero, a5=Vector3.zero, a6=Vector3.zero;
453
454         p2.x = k;
455         p2.y = j;
456         p2.z = i;
457         a=SampleGradientVolumeNew (p2);
458         //pos x-1
459         p2.x = k-1;
460         p2.y = j;
461         p2.z = i;
462         if (SampleVolume (p2) > 0.3f) {
463             contador = contador + 1;
464             a1=SampleGradientVolumeNew (p2);
465         }
466         //pos x+1
467         p2.x = k+1;
468         p2.y = j;
469         p2.z = i;
470         if (SampleVolume (p2) > 0.3f&&a1!=Vector3.zero) {
```

```
471             contador = contador + 1;
472             a2=SampleGradientVolumeNew (p2);
473         }
474         //pos y-1
475         p2.x = k;
476         p2.y = j-1;
477         p2.z = i;
478         if (SampleVolume (p2) > 0.3f) {
479             contador = contador + 1;
480             a3=SampleGradientVolumeNew (p2);
481         }
482         //pos y+1
483         p2.x = k;
484         p2.y = j+1;
485         p2.z = i;
486         if (SampleVolume (p2) > 0.3f&&a3!=Vector3.zero) {
487             contador = contador + 1;
488             a4=SampleGradientVolumeNew (p2);
489         }
490         //pos z-1
491         p2.x = k;
492         p2.y = j;
493         p2.z = i-1;
494         if (SampleVolume (p2) > 0.3f) {
495             contador = contador + 1;
496             a5=SampleGradientVolumeNew (p2);
497         }
498         //pos z+1
499         p2.x = k;
500         p2.y = j;
501         p2.z = i+1;
502         if (SampleVolume (p2) > 0.3f&&a5!=Vector3.zero) {
503             contador = contador + 1;
504             a6=SampleGradientVolumeNew (p2);
505         }
506         if (a2 == Vector3.zero&&a1 != Vector3.zero) {
507             a1 = Vector3.zero;
508             contador = contador - 1;
509         }
510         if (a4 == Vector3.zero&&a3 != Vector3.zero) {
511             a3 = Vector3.zero;
512             contador = contador - 1;
513         }
514         if (a6 == Vector3.zero&&a5 != Vector3.zero) {
515             a5 = Vector3.zero;
516             contador = contador - 1;
517         }
518         return((a + a1 + a2 + a3 + a4 + a5 + a6) / contador);
519     }
520     public Vector3 SampleGradientVolumeMeanDoble2(Vector3 p)
521     {
522         int k = (int)p.x; int j= (int)p.y; int i = (int)p.z;
523         int contador = 1;
524         Vector3 p2, a=Vector3.zero, a1=Vector3.zero, a2=Vector3.zero,
              a3=Vector3.zero, a4=Vector3.zero, a5=Vector3.zero, a6=Vector3.zero;
525
```

```
526         p2.x = k;
527         p2.y = j;
528         p2.z = i;
529         a=SampleGradientVolumeMean2 (p2);
530         //pos x-1
531         p2.x = k-1;
532         p2.y = j;
533         p2.z = i;
534         if (SampleVolume (p2) > 0.3f) {
535             contador = contador + 1;
536             a1=SampleGradientVolumeMean2 (p2);
537         }
538         //pos x+1
539         p2.x = k+1;
540         p2.y = j;
541         p2.z = i;
542         if (SampleVolume (p2) > 0.3f&&a1!=Vector3.zero) {
543             contador = contador + 1;
544             a2=SampleGradientVolumeMean2 (p2);
545         }
546         //pos y-1
547         p2.x = k;
548         p2.y = j-1;
549         p2.z = i;
550         if (SampleVolume (p2) > 0.3f) {
551             contador = contador + 1;
552             a3=SampleGradientVolumeMean2 (p2);
553         }
554         //pos y+1
555         p2.x = k;
556         p2.y = j+1;
557         p2.z = i;
558         if (SampleVolume (p2) > 0.3f&&a3!=Vector3.zero) {
559             contador = contador + 1;
560             a4=SampleGradientVolumeMean2 (p2);
561         }
562         //pos z-1
563         p2.x = k;
564         p2.y = j;
565         p2.z = i-1;
566         if (SampleVolume (p2) > 0.3f) {
567             contador = contador + 1;
568             a5=SampleGradientVolumeMean2 (p2);
569         }
570         //pos z+1
571         p2.x = k;
572         p2.y = j;
573         p2.z = i+1;
574         if (SampleVolume (p2) > 0.3f&&a5!=Vector3.zero) {
575             contador = contador + 1;
576             a6=SampleGradientVolumeMean2 (p2);
577         }
578         if (a2 == Vector3.zero&&a1 != Vector3.zero) {
579             a1 = Vector3.zero;
580             contador = contador - 1;
581         }
```

```
582         if (a4 == Vector3.zero&&a3 != Vector3.zero) {
583             a3 = Vector3.zero;
584             contador = contador - 1;
585         }
586         if (a6 == Vector3.zero&&a5 != Vector3.zero) {
587             a5 = Vector3.zero;
588             contador = contador - 1;
589         }
590         return((a + a1 + a2 + a3 + a4 + a5 + a6) / contador);
591     }
592     public Vector3 SampleGradientVolumeMean(Vector3 p)
593     {
594         int k = (int)p.x; int j= (int)p.y; int i = (int)p.z;
595         int contador = 1;
596         Vector3 p2, a=Vector3.zero, a1=Vector3.zero, a2=Vector3.zero,
              a3=Vector3.zero, a4=Vector3.zero, a5=Vector3.zero, a6=Vector3.zero;
597
598         p2.x = k;
599         p2.y = j;
600         p2.z = i;
601         a=SampleGradientVolumeNew (p2);
602         //pos x-1
603         p2.x = k-1;
604         p2.y = j;
605         p2.z = i;
606         if (SampleVolume (p2) > 0.3f) {
607             contador = contador + 1;
608             a1=SampleGradientVolumeNew (p2);
609         }
610         //pos x+1
611         p2.x = k+1;
612         p2.y = j;
613         p2.z = i;
614         if (SampleVolume (p2) > 0.3f) {
615             contador = contador + 1;
616             a2=SampleGradientVolumeNew (p2);
617         }
618         //pos y-1
619         p2.x = k;
620         p2.y = j-1;
621         p2.z = i;
622         if (SampleVolume (p2) > 0.3f) {
623             contador = contador + 1;
624             a3=SampleGradientVolumeNew (p2);
625         }
626         //pos y+1
627         p2.x = k;
628         p2.y = j+1;
629         p2.z = i;
630         if (SampleVolume (p2) > 0.3f) {
631             contador = contador + 1;
632             a4=SampleGradientVolumeNew (p2);
633         }
634         //pos z-1
635         p2.x = k;
636         p2.y = j;
```

```
637         p2.z = i-1;
638         if (SampleVolume (p2) > 0.3f) {
639             contador = contador + 1;
640             a5=SampleGradientVolumeNew (p2);
641         }
642         //pos z+1
643         p2.x = k;
644         p2.y = j;
645         p2.z = i+1;
646         if (SampleVolume (p2) > 0.3f) {
647             contador = contador + 1;
648             a6=SampleGradientVolumeNew (p2);
649         }
650         return((a + a1 + a2 + a3 + a4 + a5 + a6) / contador);
651     }
652     public Vector3 SampleGradientVolumeMeanDoble(Vector3 p)
653     {
654         int k = (int)p.x; int j= (int)p.y; int i = (int)p.z;
655         int contador = 1;
656         Vector3 p2, a=Vector3.zero, a1=Vector3.zero, a2=Vector3.zero,
              a3=Vector3.zero, a4=Vector3.zero, a5=Vector3.zero, a6=Vector3.zero;
657
658         p2.x = k;
659         p2.y = j;
660         p2.z = i;
661         a=SampleGradientVolumeMean (p2);
662         //pos x-1
663         p2.x = k-1;
664         p2.y = j;
665         p2.z = i;
666         if (SampleVolume (p2) > 0.3f) {
667             contador = contador + 1;
668             a1=SampleGradientVolumeMean (p2);
669         }
670         //pos x+1
671         p2.x = k+1;
672         p2.y = j;
673         p2.z = i;
674         if (SampleVolume (p2) > 0.3f) {
675             contador = contador + 1;
676             a2=SampleGradientVolumeMean (p2);
677         }
678         //pos y-1
679         p2.x = k;
680         p2.y = j-1;
681         p2.z = i;
682         if (SampleVolume (p2) > 0.3f) {
683             contador = contador + 1;
684             a3=SampleGradientVolumeMean (p2);
685         }
686         //pos y+1
687         p2.x = k;
688         p2.y = j+1;
689         p2.z = i;
690         if (SampleVolume (p2) > 0.3f) {
691             contador = contador + 1;
```

```
692             a4=SampleGradientVolumeMean (p2);
693         }
694         //pos z-1
695         p2.x = k;
696         p2.y = j;
697         p2.z = i-1;
698         if (SampleVolume (p2) > 0.3f) {
699             contador = contador + 1;
700             a5=SampleGradientVolumeMean (p2);
701         }
702         //pos z+1
703         p2.x = k;
704         p2.y = j;
705         p2.z = i+1;
706         if (SampleVolume (p2) > 0.3f) {
707             contador = contador + 1;
708             a6=SampleGradientVolumeMean (p2);
709         }
710         return((a + a1 + a2 + a3 + a4 + a5 + a6) / contador);
711     }
712     public Vector3 SampleGradientVolumeMatrix(Vector3 p)
713     {
714         int k = (int)p.x; int j= (int)p.y; int i = (int)p.z;
715         int contador = 0;
716         Vector3 p2, a=Vector3.zero;
717         p2.x = k;
718         p2.y = j;
719         p2.z = i;
720         for (int ii = -1; ii < 2; ++ii) {
721             for (int jj = -1; jj < 2; ++jj) {
722                 for (int kk = -1; kk < 2; ++kk) {
723                     p2.x = k + kk;
724                     p2.y = j + jj;
725                     p2.z = i + ii;
726                     if (SampleVolume (p2) > 0.3f) {
727                         if (Math.Abs (ii + jj + kk) == 3) {
728                             contador = contador + 1;
729                             a = a + SampleGradientVolumeNew (p2);
730                         }
731                         if (Math.Abs (ii + jj + kk) == 2) {
732                             contador = contador + 2;
733                             a = a + SampleGradientVolumeNew (p2) * 2;
734                         }
735                         if (Math.Abs (ii + jj + kk) == 1) {
736                             contador = contador + 3;
737                             a = a + SampleGradientVolumeNew (p2) * 3;
738                         }
739                         if (Math.Abs (ii + jj + kk) == 0) {
740                             contador = contador + 5;
741                             a = a + SampleGradientVolumeNew (p2) * 5;
742                         }
743                     }
744                 }
745             }
746         }
747
```

```
748         return a  / (contador);
749     }
750     #endregion
751 
752     /// <summary>
753     /// Properly creates a <see cref="Volume"/> instance.
754     /// </summary>
755     /// <returns>New empty <see cref="Volume"/> object</returns>
756     public static Volume CreateVolume()
757     {
758         Volume volume = ScriptableObject.CreateInstance<Volume>();
759         return volume;
760     }
761 
762     private void MakePreview()
763     {
764         int resx, resy, resz;
765         resx = volumeTexture.width;
766         resy = volumeTexture.height;
767         resz = volumeTexture.depth;
768 
769         int scale = Math.Max(resx, resy) / maxPreviewSize;
770 
771         _previewTexture = new Texture2D(volumeTexture.width / scale,
              volumeTexture.height / scale, TextureFormat.Alpha8, false);
772 
773         for (int i = 0; i < resx; i += scale)
774         {
775             for (int j = 0; j < resy; j += scale)
776             {
777                 float accumulator = 0.0f;
778                 for (int k = 0; k < resz; k += scale)
779                 {
780                     long pixelPos = k * resx * resy + j * resx + i;
781                     accumulator += pixels[pixelPos].a / 255.0f;
782                 }
783                 float alpha = accumulator / (float)(resz / scale);
784                 _previewTexture.SetPixel(i / scale, j / scale, new Color(0, 0,
                      0, alpha));
785             }
786         }
787 
788         _previewTexture.Apply();
789     }
790 
791     /// <summary>
792     /// Properly updates a <see cref="Volume"/> asset stored at <paramref
          name="assetPath"/>.
793     /// </summary>
794     /// <param name="assetPath">Path of the currently existing asset to be
          updated</param>
795     public void UpdateAsset(string assetPath)
796     {
797         Texture3D texture = volumeTexture;
798         if (texture != null)
799         {
```

```
800             texture.name = "Volume Texture";
801             texture.hideFlags = HideFlags.HideInHierarchy;
802
803
804             AssetDatabase.AddObjectToAsset(texture, this);
805             AssetDatabase.ImportAsset(assetPath);
806
807             EditorUtility.SetDirty(texture);
808             EditorUtility.SetDirty(this);
809             AssetDatabase.SaveAssets();
810             AssetDatabase.Refresh();
811
812             AssetDatabase.SaveAssets();
813         }
814
815         texture = gradientTexture;
816         if (texture != null)
817         {
818             texture.name = "Gradient Texture";
819             texture.hideFlags = HideFlags.HideInHierarchy;
820
821             AssetDatabase.AddObjectToAsset(texture, this);
822             AssetDatabase.ImportAsset(assetPath);
823
824             EditorUtility.SetDirty(texture);
825             EditorUtility.SetDirty(this);
826             AssetDatabase.SaveAssets();
827             AssetDatabase.Refresh();
828
829             AssetDatabase.SaveAssets();
830         }
831
832         Texture2D previewTexture = _previewTexture;
833         if (previewTexture != null)
834         {
835             previewTexture.name = "Preview Texture";
836             previewTexture.hideFlags = HideFlags.HideInHierarchy;
837
838             AssetDatabase.AddObjectToAsset(previewTexture, this);
839             AssetDatabase.ImportAsset(assetPath);
840
841             EditorUtility.SetDirty(previewTexture);
842             EditorUtility.SetDirty(this);
843             AssetDatabase.SaveAssets();
844             AssetDatabase.Refresh();
845
846             AssetDatabase.SaveAssets();
847         }
848     }
849     //proves
850
851 }
852
853
```

Script HapticContactMed

F:\VRMedCode\HapticContactMed.cs 1

```
1  using UnityEngine;
2  using System.Collections;
3  using UnityEngine.UI;
4  using System.Collections.Generic;
5  public class HapticContactMed : MonoBehaviour
6  {
7      public VolumeRenderer vr;
8      public GameObject cil;
9
10     float timeForce;
11
12     public Transform CropTransform;
13     //per veure normal
14     public GameObject PaintedObjects;
15
16     List<GameObject> ListVox = new List<GameObject>();
17
18
19     Vector3 mitjanavec=Vector3.zero;
20     bool con=false;
21     Vector3[] prova= new Vector3[5];
22
23     // Use this for initialization
24     void Start()
25     {
26         for (int i = 1; i < prova.Length; i++) {
27             prova[i]=Vector3.zero;
28         }
29
30     }
31
32     // Update is called once per frame
33     void LateUpdate()
34     {
35
36
37
38         timeForce += Time.deltaTime;
39         /*if (Menu_Haptic.currentState == "Force" ) {
40             if (con==false) {
41                 timeForce = 0;
42             }
43             if (vr.SampleVolume (transform.position) > 0.3f) {
44                 Force ();
45             }
46         }*/
47
48         if (Menu_Haptic.currentState == "Force" ) {
49
50
51             ApplyContactForce ();
52
53             //CreateVoxelsTouchable();
54
55
56         }
```

```
57          else if (Menu_Haptic.currentState == "Touch" && timeForce>0.1f ) {
58              timeForce = 0;
59              CreateVoxelsTouchable ();
60          }
61
62          if (Menu_Haptic.currentState != "Touch") {
63              for(int i = 0; i < ListVox.Count; i++)
64              {
65                  Destroy(ListVox[i].gameObject);
66              }
67
68              ListVox.Clear();
69          }
70
71
72      }
73
74      void ApplyContactForce()
75      {
76
77
78          float magnitude = vr.SampleVolume(transform.position);
79
80          //afegit per mi
81          if (vr.SampleVolume(transform.position) > 0.3f && timeForce > 0.5f)
82          {
83              //lastgrad = 3 * lastgrad + grad;
84              //lastgrad /= 4;
85              timeForce = 0;
86              Vector3 grad = vr.SampleGradient3(transform.position);
87              float[] directionEffect = new float[3];
88              directionEffect[0] = grad.x;
89              directionEffect[1] = grad.y;
90              directionEffect[2] = -grad.z;
91              GameObject PaintVox = (GameObject)Instantiate
                  (Resources.Load<GameObject> ("PaintVox"), transform.position,
                  Quaternion.identity, PaintedObjects.transform);
92              GameObject.Destroy(PaintVox, 35.0f);
93              for (int i = 1; i < 20; i++) {
94                  GameObject PaintVox2 = (GameObject)Instantiate
                      (Resources.Load<GameObject> ("PaintVox2"),
                      transform.position + 0.001f * grad*i, Quaternion.identity,
                      PaintedObjects.transform);
95                  GameObject.Destroy(PaintVox2, 35.0f);
96              }
97              /*for (int i = 1; i < 20; i++) {
98                  GameObject PaintVox3 = (GameObject)Instantiate
                      (Resources.Load<GameObject> ("PaintVox3"),
                      transform.position+(transform.position -vr.Vectorcercle())
                      *0.02f *i, Quaternion.identity, PaintedObjects.transform);
99                  GameObject.Destroy(PaintVox3, 35.0f);
100             }*/
101             //DrawLine (transform.position, -grad*vr.volume.voxelSize.x*0.05f
                  +transform.position, Color.green,100.0f);
102             Vector3 teoric=(transform.position - vr.Vectorcercle
                  ()).normalized;
```

```
103             Vector3 dif = (grad.normalized - teoric);
104             float prodesc = Vector3.Dot (grad.normalized, (transform.position
                  - vr.Vectorcercle ()).normalized);
105             float angle = Mathf.Acos (prodesc)*180/Mathf.PI;
106             //print ("Calculat:"+grad.normalized+" Teòric"+teoric+" Diferència
                   vectors normalitzats:"+dif+" Producte escalar:"+prodesc+" Angle
                   de diferència:"+angle);
107             PluginImport.SetEffect(ConverterClass.ConvertStringToByteToIntPtr
                  ("constant"), 0, 0, magnitude/2, 200, 0,
                  ConverterClass.ConvertFloat3ToIntPtr(directionEffect),
                  ConverterClass.ConvertFloat3ToIntPtr(directionEffect));
108             PluginImport.StartEffect(0);
109
110         }
111
112
113     }
114     //prova
115     void ApplyContactForce2(Vector3 mitjana)
116     {
117
118
119         float magnitude = vr.SampleVolume(transform.position);
120
121         //afegit per mi
122         if (vr.SampleVolume(transform.position) > 0.3f )
123         {
124             //lastgrad = 3 * lastgrad + grad;
125             //lastgrad /= 4;
126
127             float[] directionEffect = new float[3];
128             directionEffect[0] = mitjana.x;
129             directionEffect[1] = mitjana.y;
130             directionEffect[2] = -mitjana.z;
131             GameObject PaintVox = (GameObject)Instantiate
                  (Resources.Load<GameObject> ("PaintVox"), transform.position,
                  Quaternion.identity, PaintedObjects.transform);
132             GameObject.Destroy(PaintVox, 15.0f);
133             for (int i = 1; i < 20; i++) {
134                 GameObject PaintVox2 = (GameObject)Instantiate
                      (Resources.Load<GameObject> ("PaintVox2"),
                      transform.position + 0.005f * mitjana*i,
                      Quaternion.identity, PaintedObjects.transform);
135                 GameObject.Destroy(PaintVox2, 15.0f);
136             }
137             for (int i = 1; i < 20; i++) {
138                 GameObject PaintVox3 = (GameObject)Instantiate
                      (Resources.Load<GameObject> ("PaintVox3"),
                      transform.position+(transform.position -vr.Vectorcercle())
                      *0.02f *i, Quaternion.identity, PaintedObjects.transform);
139                 GameObject.Destroy(PaintVox3, 15.0f);
140             }
141             //DrawLine (transform.position, -grad*vr.volume.voxelSize.x*0.05f
                  +transform.position, Color.green,100.0f);
142             if(magnitude>0.8){magnitude=0.8f;}
143             Vector3 teoric=(transform.position - vr.Vectorcercle
```

```
                ()).normalized;
144             Vector3 dif = (mitjana.normalized - teoric);
145             float prodesc = Vector3.Dot (mitjana.normalized,
                  (transform.position - vr.Vectorcercle ()).normalized);
146             float angle = Mathf.Acos (prodesc)*180/Mathf.PI;
147             print ("Calculat:"+mitjana.normalized+" Teòric"+teoric+"
                  Diferència vectors normalitzats:"+dif+" Producte
                  escalar:"+prodesc+" Angle de diferència:"+angle);
148             PluginImport.SetEffect(ConverterClass.ConvertStringToByteToIntPtr
                  ("constant"), 3, 0, magnitude/2, 200, 0,
                  ConverterClass.ConvertFloat3ToIntPtr(directionEffect),
                  ConverterClass.ConvertFloat3ToIntPtr(directionEffect));
149             PluginImport.StartEffect(3);
150
151         }
152
153         con = false;
154         mitjanavec = Vector3.zero;
155         for (int i = 1; i < prova.Length; i++) {
156             prova[i]=Vector3.zero;
157         }
158     }
159
160     void CreateVoxelsTouchable() {
161         #region generar vox
162         if (ListVox.Count == 0)
163         {
164             for (int x = 0; x < 5 * 5 * 5; x++)
165             {
166
167
168                 GameObject Vox = (GameObject)Instantiate
                      (Resources.Load<GameObject>("Vox"), Vector3.up,
                      Quaternion.identity);
169                 //Vox.transform.SetParent(CropTransform, true);
170                 Vox.transform.SetParent(GameObject.Find("Voxs").transform,
                      true);
171                 ListVox.Add(Vox);
172
173
174
175             }
176             GameObject.Find("dummy").GetComponent<SimpleShapeContact>
                  ().OnDisable();
177             GameObject.Find("dummy").GetComponent<SimpleShapeContact>().Start
                  ();
178         }
179         #endregion;
180
181         if (!PluginImport.GetButton2State())
182         {
183
184             for (int x = -2; x < 3; x++)
185             {
186
187                 for (int y = -2; y < 3; y++)
```

```
188                 {
189
190                     for (int z = -2; z < 3; z++)
191                     {
192                         Vector3 pos = transform.position + new Vector3(x, y,
                    z) / 200;
193                         pos *= 200;
194                         pos = new Vector3(Mathf.Floor(pos.x), Mathf.Floor
                    (pos.y), Mathf.Floor(pos.z));
195                         pos /= 200;
196
197                         if (vr.SampleVolume(pos) > 0.3f)
198                         {
199                             GameObject Vox = ListVox[(x + 2) + 5 * (y + 2) +
                    5*5 * (z + 2)];
200                             Vector3 grad = vr.SampleGradient2
                    (transform.position);
201                             float[] al = new float[3];
202                             al[0] = -grad.x;
203                             al[1] = -grad.y;
204                             al[2] = -grad.z;
205                             Vector3 alfa=new Vector3();
206                             alfa.x = -Mathf.Atan (al[1]/al[2])*180/Mathf.PI;
207                             alfa.y = Mathf.Atan (al[0]/al[2])*180/Mathf.PI;
208                             alfa.z = 0;
209                             Vox.transform.eulerAngles = alfa;
210                             Vox.transform.localScale = (new Vector3(0.02f,
                    0.02f, 1)).DividedBy(CropTransform.localScale);
211                             bool exist=false;
212
213                             foreach (GameObject obj in ListVox) {
214                                 if (pos == obj.transform.position) {
215                                     exist = true;
216                                 }
217                             }
218                             if (!exist)
219                             {
220
221                                 Vox.transform.position = pos;
222
223                             }
224
225                         }
226                     }
227                 }
228             }
229
230             /*if (vr.SampleVolume (transform.position) > 0.3f) {
231                 float[] directionEffect = new float[3];
232                 directionEffect [0] = 0;
233                 directionEffect [1] = 0;
234                 directionEffect [2] = 0;
235
236
237                 PluginImport.SetEffect
                      (ConverterClass.ConvertStringToByteToIntPtr ("friction"), 5,
```

```
                    0.4f, 1f, 300f, 0, ConverterClass.ConvertFloat3ToIntPtr
                   (directionEffect), ConverterClass.ConvertFloat3ToIntPtr
                   (directionEffect));
238                PluginImport.StartEffect (5);
239            }*/
240
241        }
242
243        else
244
245        {
246            for (int i = 0; i < 3 * 3 * 3; i++)
247            {
248                GameObject Vox = ListVox[i];
249                Vox.transform.position = Vector3.up;
250            }
251
252        }
253    }
254    void DrawLine(Vector3 start, Vector3 end, Color color, float duration =
         0.2f)
255    {
256        GameObject myLine = new GameObject();
257        myLine.name = "Line";
258        myLine.transform.position = start;
259        myLine.AddComponent<LineRenderer>();
260        LineRenderer lr = myLine.GetComponent<LineRenderer>();
261        lr.material = new Material(Shader.Find("Particles/Alpha Blended
           Premultiply"));
262        lr.SetColors(color, color);
263        lr.SetWidth(0.005f, 0.005f);
264        lr.SetPosition(0, start);
265        lr.SetPosition(1, end);
266        GameObject.Destroy(myLine, duration);
267    }
268    void Force(){
269        float[] magnitud= new float[5];
270        con = true;
271        if (timeForce < 0.04f&&vr.SampleVolume (transform.position)>0.3f) {
272            prova [0] = vr.SampleGradient3 (transform.position);
273            magnitud [0] = vr.SampleVolume (transform.position);
274        } else if (timeForce < 0.08f&&vr.SampleVolume (transform.position)
             >0.3f) {
275            prova [1] = vr.SampleGradient3 (transform.position);
276            magnitud [1] = vr.SampleVolume (transform.position);
277        } else if (timeForce < 0.12f&&vr.SampleVolume (transform.position)
             >0.3f) {
278            prova [2] = vr.SampleGradient3 (transform.position);
279            magnitud [2] = vr.SampleVolume (transform.position);
280        } else if (timeForce < 0.16f&&vr.SampleVolume (transform.position)
             >0.3f) {
281            prova [3] = vr.SampleGradient3 (transform.position);
282            magnitud [3] = vr.SampleVolume (transform.position);
283        } else if (timeForce < 0.2f&&vr.SampleVolume (transform.position)
             >0.3f) {
284            prova [4] = vr.SampleGradient3 (transform.position);
```

```
285             magnitud [4] = vr.SampleVolume (transform.position);
286         }else {
287             timeForce = 0;
288             float mitjanamag = 0;
289             for (int i = 1; i < prova.Length; i++) {
290                 mitjanavec += prova [i];
291                 mitjanamag += magnitud [i];
292             }
293             mitjanavec.Normalize ();
294             ApplyContactForce2 (mitjanavec);
295         }
296     }
297 }
298
```