ISDN5300 - Final Report
Sketching 3D animation for Real Indoor Environment

ZHENG Kexin 20970926
HUANG Haoming 21036105

1 Introduction

This project aims to develop an interactive system for creating 3D avatar animation easily given a scanned 3D environment. The key challenges are twofold. Firstly, how to process the scanned 3D scene to extract well-define geometry that can support animation creation. Secondly, how to create the animation with simple user inputs to provide good user experience.
To tackle the aforementioned challenges, we developed a framework with two key functionalities: 3D scene understanding and sketch-based animation interface. The 3D reconstruction starts from the traditional TSDF integration from the RGBD data and segmentation is applied to produce instance-wise pointclouds and meshes. The 3D scene understanding Sketch-based interface is inspired by MotionDoodles[9]. Taken 2D sketch as input, the system can transform it into a 3D trajectory with underlying scene geometry and assign corresponding animation preset to generate the final animation.
In summary the objectives of the project are:

1. Develop a 3D scene understanding scheme to process raw scanned 3D scene data for animation creation.

2. Develop a sketch-based animation interface to composite avatar motions with simple 2D sketch input.

3. Develop an interactive system to create 3D avatar animation with scanned real scene data.

2 System Design

The system framework is shown in Figure 1.

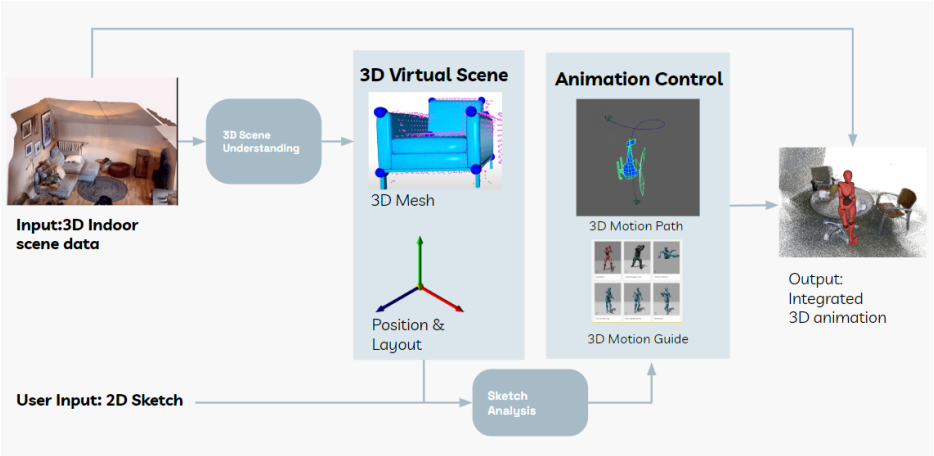


Figure 1: Proposed Framework

3 Methodology

3.1 3D scene understanding

3.1.1 Dense Mapping

The 3D scene reconstruction module uses RGBD and pose data and integrates a TSDF (Truncated Signed Distance Function) volume to create a detailed 3D representation. The Truncated Signed Distance Function (TSDF) is a methodological framework utilized in 3D reconstruction from depth data[2], which operates by assigning a signed distance value to each point in a 3D space relative to the nearest surface of an object; positive outside the surface, negative inside, and zero precisely at the surface. The main function processes individual scene directories, handling the reading of camera intrinsics, and initializing a TSDF volume. Within this function, there's a loop that iterates over each frame of the scan data, integrating RGB and depth images along with camera poses into the TSDF volume, as shown in Figure.2. After processing all the frames, the script extracts a mesh from the TSDF volume and saves it, resulting in a 3D reconstruction of the scene.

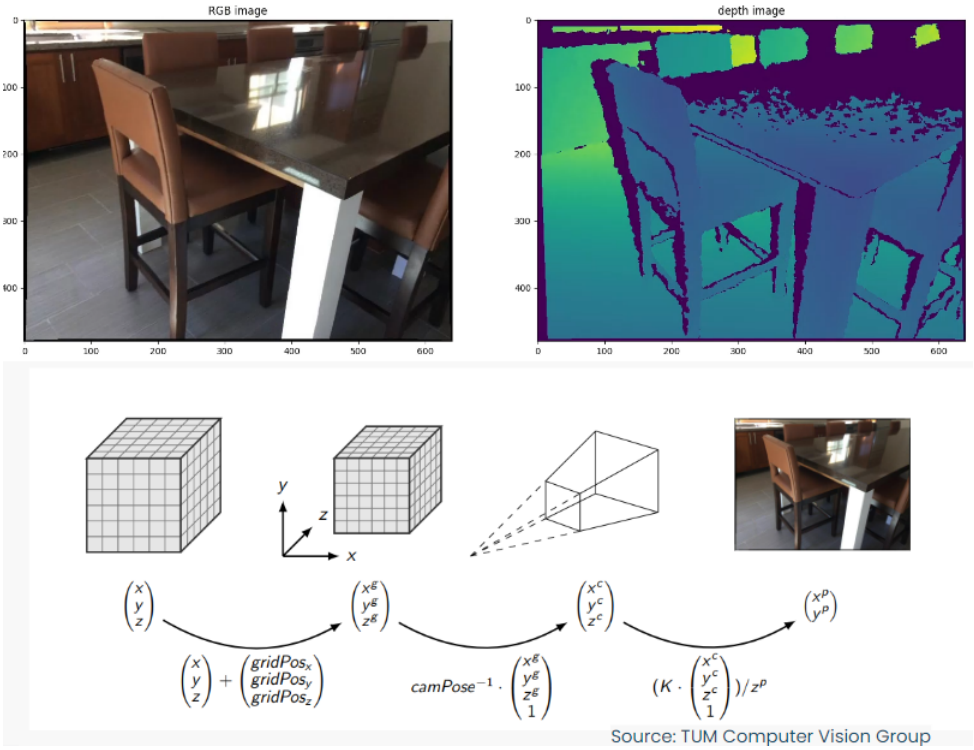


Figure 2: TSDF integration for each frame

In the dense mapping process, TSDF amalgamates depth data from multiple viewpoints, incrementally updating the voxel values in the grid. Subsequently, the surface of the reconstructed object is delineated by extracting the zero-level set of the TSDF grid, employing the Marching Cubes algorithms[8], as shown in Figure. 3.

3.1.2 3D Geometric Segmentation

The geometric segmentation of 3D point cloud data employs an octree-based segmentation. The resulting segmented data, including a labeled RGB point cloud and a ground cloud, are extracted from the Octree object. The segmentation process includes splitting non-leaf nodes into smaller ones, recursively detecting planes in each, and merging nodes that belong to the same plane based on their orientation and proximity, as shown in Figure. 4.

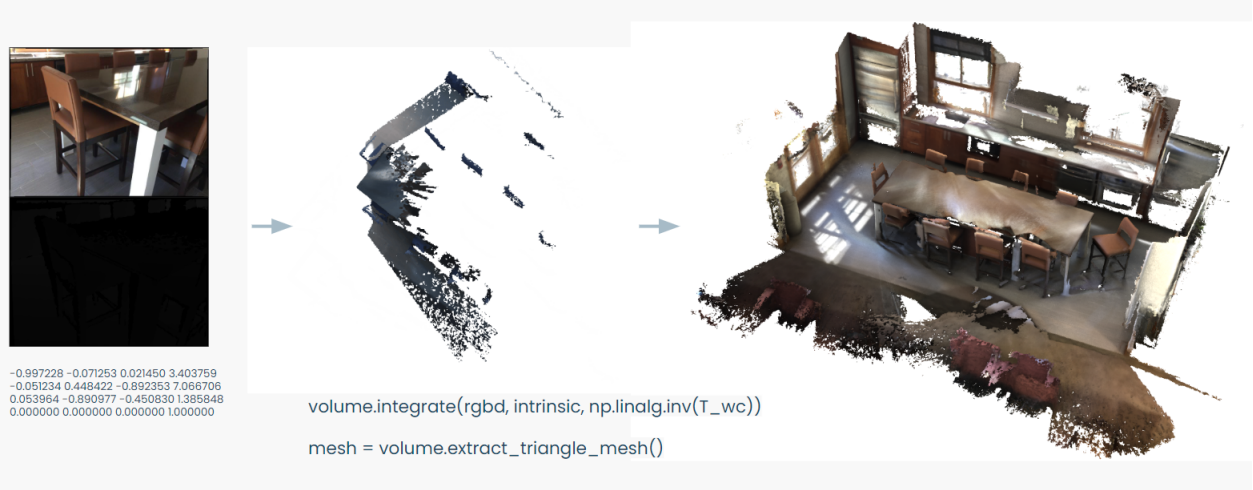


Figure 3: Dense mapping process

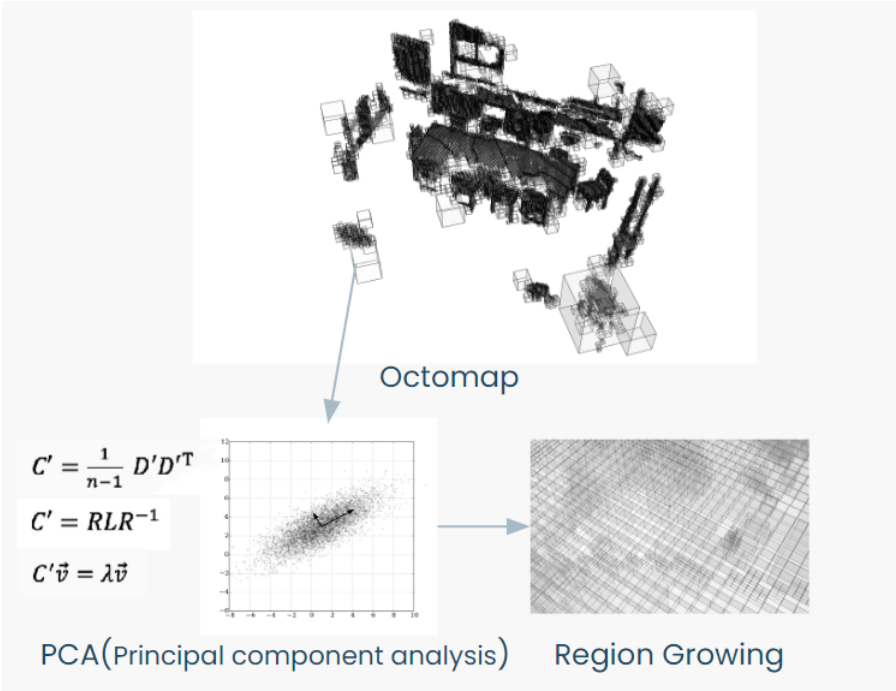


Figure 4: Octree-based segmentation

This results in the segmentation of the point cloud into distinct planar regions, categorizing the spatial data into segments, as shown in Figure. 5.

3.1.3 Instance Segmentation aided by foundation model predictions

The segmentation uses prediction from the foundation model (Grounded-SAM)[1]. The implementation of Grounded-SAM would be discussed more in the Limitation and Future work section. The preparation of segmentation performs Non-Maximum Suppression (NMS) by sorting detections based on their scores, computing Intersection over Union (IoU) for pairs of detections, and eliminating those that surpass a defined IoU threshold, indicative of redundancy. We employ this method to filter a list of detections. It calculates a score inversely related to each detection's mask area, hypothesizing that smaller detections

[1]https://github.com/IDEA-Research/Grounded-Segment-Anything

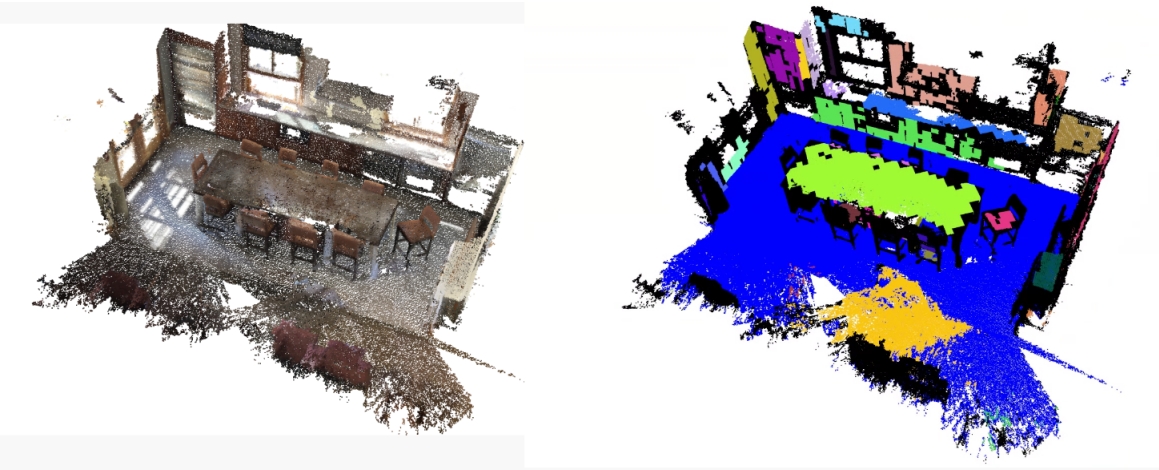


Figure 5: Plane segmentation results

might be more likely to overlap, and then applies NMS. The outcome is a refined set of detections where redundancies are minimized, enhancing the accuracy and efficiency of the detection process in scenarios with multiple detections of the same object., as shown in Figure. 6.

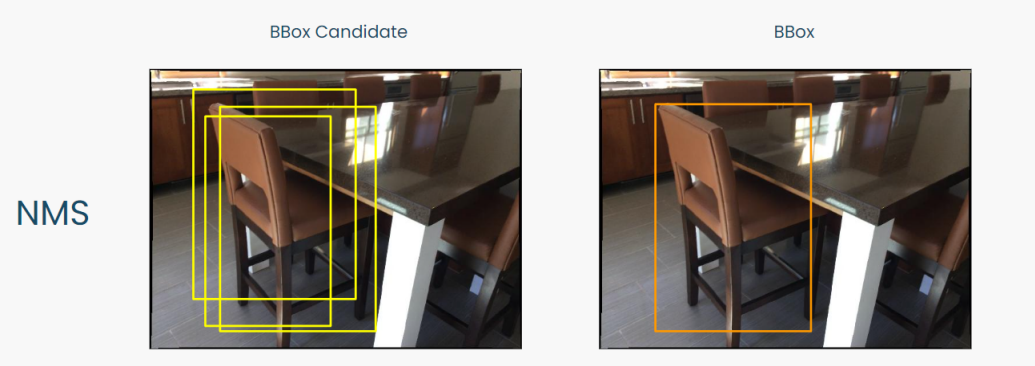


Figure 6: Non-Max Suppression

The instance segmentation module revolves around the InstanceMap class, which manages semantic instances within a 3D scene. This class tracks instances, merges overlapping ones, and updates voxel points within the map. The process involves loading RGB-D frames, generating point clouds, and assigning prediction labels to these clouds. It achieves this by computing the IoU between each detection's mask and the projection map of active instances. The function creates an IoU matrix to measure overlaps and an assignment matrix to determine the best match based on IoU values, adhering to a specified minimum IoU threshold. It addresses the challenge of multiple detections being assigned to a single instance by ensuring each detection is matched with only one instance, prioritizing the detection with the highest IoU, as shown in Figure. 7.

3.1.4 Objects Simplification

The generation of objects begins by loading a point cloud from a file, which is then voxelized using Open3D to create a grid of cubes, each representing a section of the point cloud space. For each voxel, a cube mesh is generated with dimensions matching the voxel size. These cube meshes are converted to Trimesh objects and are subsequently merged into a single mesh using Trimesh's boolean union function. Then the voxel mesh is imported into Rhino to reduce the faces of mesh, and then imported into Unity as shown in Figure. 8.

Figure 7: Association and integration of instances

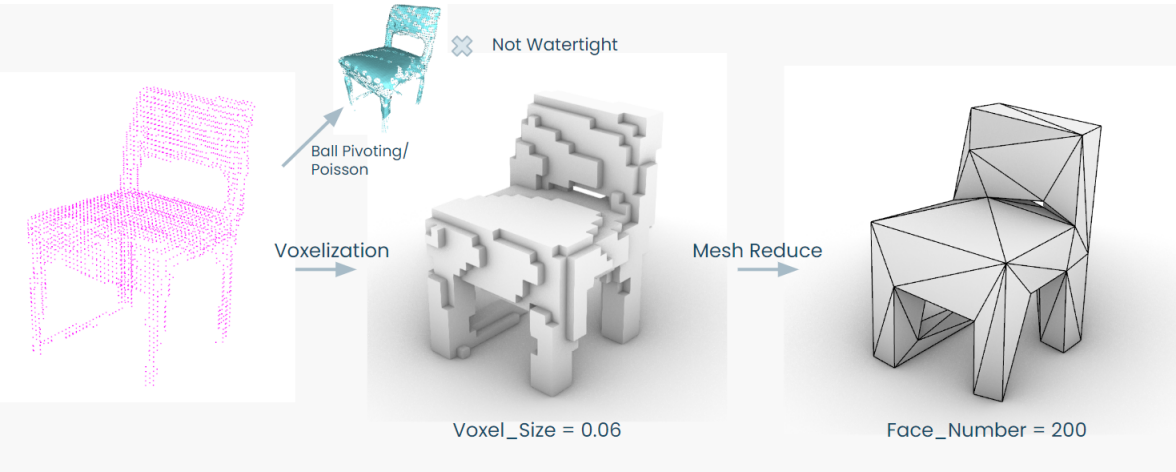


Figure 8: Simplified Mesh

3.2 Sketch-based animation creation

The sketch-based animation creation consists of three parts: Motion sketch recognition, scene-based 3D trajectory creation and animation composition. The input 2D sketch is recognized and translated into corresponding motions. The 3D trajectory is then created using the scene geometry obtained from the 3D scene understanding scheme. Subsequently, the motion and 3D trajectory are temporally aligned to produce the final animation. We use animation presets from the Mixamo [2] as the base motion clips. The pipeline is illustrated in Figure. 9.

3.2.1 Motion Sketch Recognition

The system incorporates the motion sketching concept introduced by Motion Doodles [9], which translates a set of sketches into various motions. The current system supports 6 different kinds of motion. The motions are selected based on a user study conducted in AR animator[12], which identified motions that are most commonly chosen by users for creating in-context avatar animations, which aligns with our target scenes. The detailed motions and corresponding sketches are shown in Figure. 10.

The sketch points are first segmented in to small segments before recognition. Motion Doodles[9]

[2]www.mixamo.com

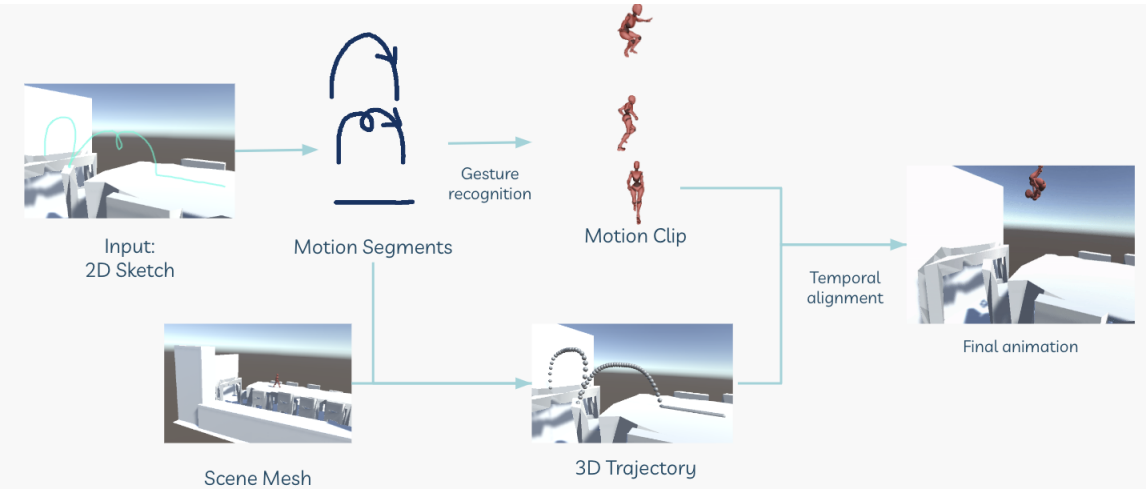


Figure 9: Sketch-based animation creation pipeline

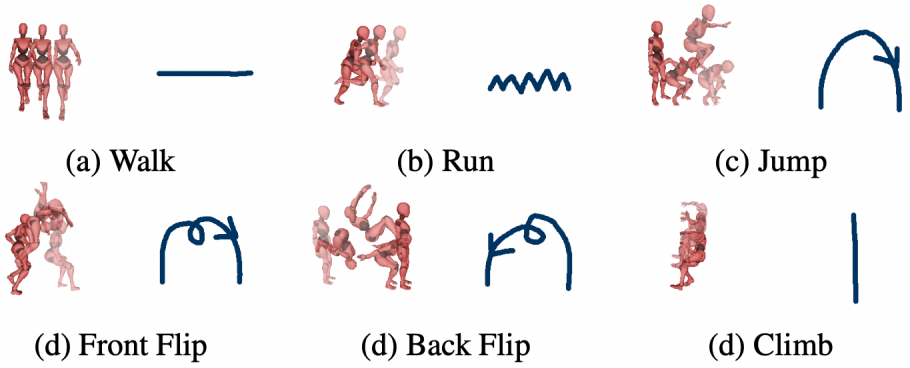


Figure 10: Motion sketches and corresponding motion

employed a purely 2D approach to perform segmentation, assuming that all motions are co-planar. Token segmentation is performed based on changes in vertical movements and corner detection. The tokens are then used for interpreting motions with pre-defined motion dictionary. The corner detection algorithm[1] they used requires parameter tuning to achieve optimal performance. Such approach is prone to errors and limited the sketch orientation. As illustrated in Figure. 11, the highlighted segmented point should not be detected. Considering our system is based on mouse interaction and mouse release is a simple

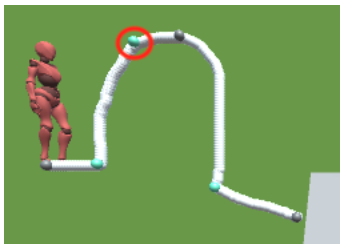


Figure 11: Error segmentation using original motion doodle method. The point highlighted in red should not be a token segment point

action to perform. We record the motion segmentation points as mouse release positions.

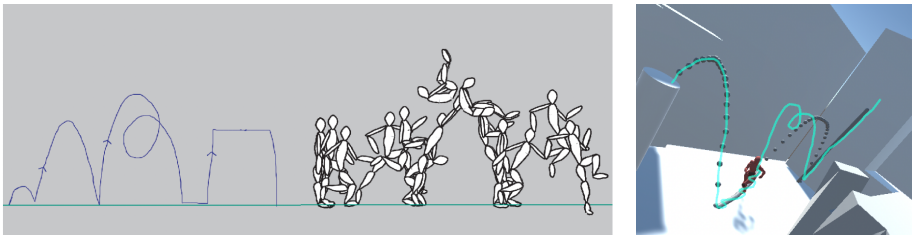


Figure 12: In comparison with the original motion doodle(Left), the sketch support by our system(Right) is not limited by the drawing orientation and different motions can lie on different planes.

After obtaining the motion segments, they are fed into $1 recognizer[10] for template-based motion recognition. This recognizer is invariant to position, rotation and scale[10], allowing for greater user freedom during sketching. To facilitate the addition of a wider variety of motion into the system, we developed a simple interface for saving motion sketch templates as XML files. Since $1 recognizer struggles with straight line segment due to the uniform scaling [10]performed during the recognition process, we use the arc length to end point distance ratio to detect straight line segments before input it into the recognizer. Additionally, we distinguish between 'walk' and 'climb' motion based on vertical position changes in the 3D trajectory.
In comparison with the original approach(See Figure.12), the motion segmentation and recognition adopted in the current system offer greater flexibility for users to sketch different motions, and improves recognition accuracy.

3.2.2 Scene-based 3D trajectory Creation

Initially, we proposed using Skippy[6] to generate 3D trajectories from 2D sketches. However, upon observing and experimenting with the processed scene data, we discovered that trajectory points within a single motion are typically co-planar. Thus, we designed a 3D trajectory creation method based on scene geometry and motion types, utilizing the co-planar assumption.
For motions where the avatar has close contact with the underlying scene, such as 'walk', 'run', and 'climb', the trajectory is generated by raycasting the sketch points from screen space to the scene geometry. Since the 'run' motion sketch is a polyline, a central path is estimated from the input sketch points using midpoint smoothing and Douglas–Peucker algorithm[3] prior to performing the raycasting.
When creating the trajectory for in-air motions, only the two end points are raycasted to find corresponding 3D positions. We assume the trajectory points lies on a plane formed by the two 3D end points p_1 and p_2, and the up vector $v_{up} = \begin{bmatrix}0 & 1 & 0\end{bmatrix}$. The normal n can be computed as

$$n = \frac{v_{up} \times (p1 - p2)}{||v_{up} \times (p1 - p2)||} \tag{1}$$

Form a ray from the camera r_o to the sketch point with direction r_d. Let the distance from the intersection point $\hat{p}$ is λ.

$$\begin{aligned} ((r_o + \lambda r_d) - p_1) \cdot n = 0 \\ \lambda = \frac{p_1 \cdot n - r_o \cdot n}{r_d \cdot n} \end{aligned} \tag{2}$$

The final 3D points on the 3D trajectory can be obtained by $\hat{p} = r_o + \lambda r_d$. To simulate a realistic jump with the influence of gravity, the number of frames for the ascending and descending phases are computed based on the height differences between the two endpoints and the highest point. This calculation takes into account the gravity equation, which is expressed as $h = \frac{1}{2}gt^2$. We set a maximum frame number empirically as $t_{max} = 25$ frames and a max jump height as $h_{max} = 2.5f$. The frame number of phases t

can be computed as

$$t = \sqrt{\frac{h}{h_{max}}} t_{max} \tag{3}$$

After computed the frame number, the height of each points on the trajectory can be computed following the gravity equation. We found point on the existing sketch points that has the closest height as the final 3D position. This results in points become closer near the highest point and becomes further apart when reaching ground, as depicted in Figure.13. Similar to the 'run' sketch, the shape 'front flip' sketch and

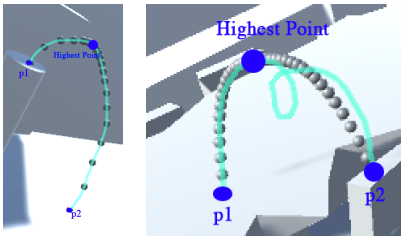


Figure 13: Generated in-air motion 3D trajectories. Jump(Left). Flip(Right)

'back flip' sketch let them cannot be directly used to generate the 3D trajectory. Thus the height of the sketch is considered. Thus, the height of the sketch is taken into consideration. Points are evenly sampled along the movements on the x-axis and z-axis from the raycasted start point to the end point. The y-value of each points is computed in a similar way as the 'jump' motion.

3.2.3 Animation Composition

The generated 3D trajectory and default motion clips have different lengths. For the 3D trajectory, each point is considered as one frame. For repetitive motions, including 'walk', 'run', and 'climb', the keyframes are repeated along the trajectory points. The situation for in-air animation is more complex. As illustrated in Figure.14, the in-air motions usually consists of multiple states and the position of avatar should be controlled accordingly. As a results, we manually defined the keyframe of different states from the base motion clip.The 'Anticipate', 'Launch', 'Landing', and 'Followthrough' states are performed at fixed end point positions. The keyframes of 'Ascent' and 'Descent' states are interpolated for each position among the 3D trajectory.

In addition to the position and joint rotation, the orientation of the avatar is also essential for creating

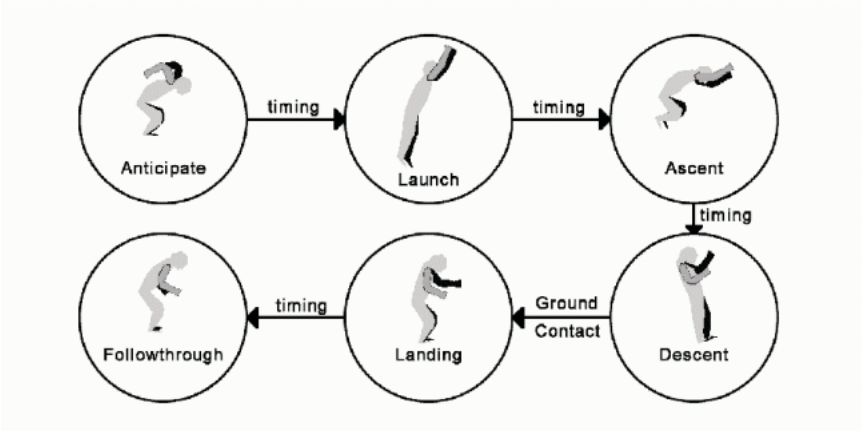


Figure 14: Different state of in-air motion. Image Credit: Motion Doodles[9]

natural 3D animation. The orientations are computed based on different motion types. For 'walk' and

'run', the facing orientation is computed as cross product of the tangent of trajectory and the up vector. For 'climb' motion, the orientation is the inverse of the plane normal on which the avatar lies. For in-air motion, the face orientation is the cross product of the up vector and the movement along the x-axis and z-axis.
After obtaining the position, joint rotation, and avatar orientation, the system display the animation at a frame rate of 30 frames per seconds.

3.3 User Interface

The interface of the final system is shown in Figure.15. The user can start drawing the sketch by clicking on the 'Draw Line' button. After finish drawing for one motion, the user can release the button and the system will record the release point for sketch segmentation. The user can repeat the draw and release action to combine different motions. Finally, user can press the 'Play Animation' button to view the created animation. To view the animation with scanned scene data, the user can check the 'Show Real Scene' toggle.
The user can navigate the scene with mouse-based camera control. They can move forward and backward by scrolling, and move left and right by pressing on the scroll wheel while moving the mouse. The camera rotation is controlled by left mouse click and drag.

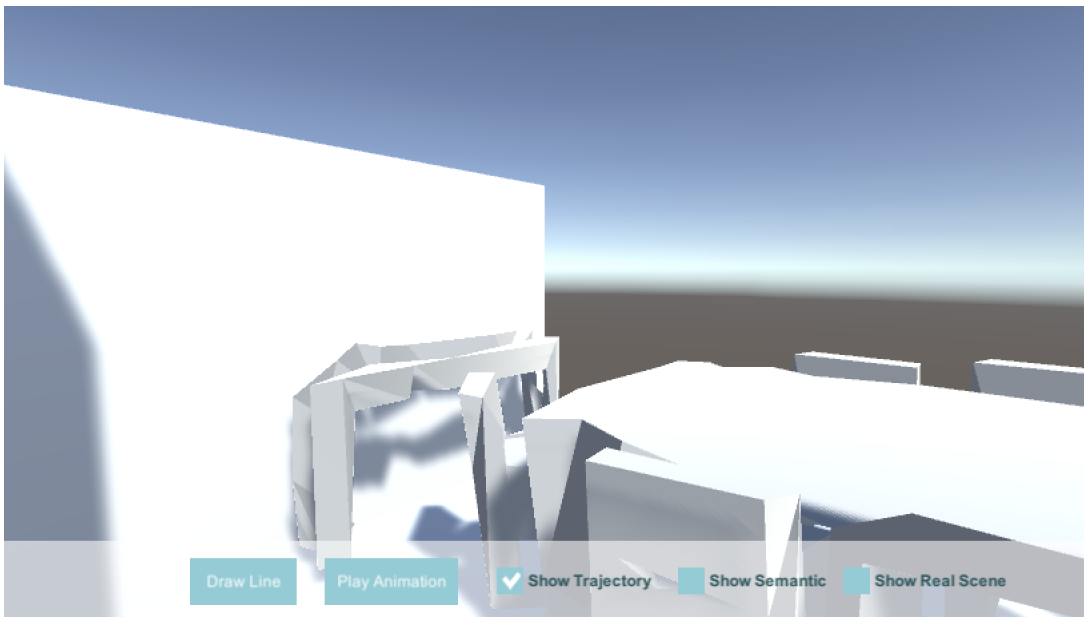


Figure 15: System user interface

4 Results

4.1 Demo Video

The demo video can be viewed from this link.

4.2 More results

More animations created with our system can be viewed from this folder.

5 Implementation

The 3D scene understanding is developed using python. The extracted 3D scene data becomes the input for the sketch-based animation interface developed with Unity coded with C sharp.

5.1 3D scene understanding

The code for dense volumetric fusion using Open3d on ScanNet dataset is in the ”scripts”folder.
dense_mapping.py:The core of the function is a loop that iterates through depth frames of the scene. For each frame, it checks if the frame aligns with the specified frame gap to manage the density of the processed data. The function then loads the corresponding RGB and depth images, along with the camera pose. These are used to create an RGB-D image, which is then transformed into a point cloud. This point cloud is integrated into the TSDF volume, progressively building a detailed 3D model of the scene.

```
# Init
intrinsic = o3d.camera.PinholeCameraIntrinsic()
intrinsic.set_intrinsics(depth_dim[1],depth_dim[0],K_depth[0,0],K_depth[1,1],K_depth[0,2],K_depth[1,2])

volume = o3d.pipelines.integration.ScalableTSDFVolume(
    voxel_length=4.0 / resolution,
    sdf_trunc=SDF_TRUCT,
    color_type=o3d.pipelines.integration.TSDFVolumeColorType.RGB8)

#
tmp_dir = os.path.join(scene_dir,'tmp')
if os.path.exists(tmp_dir)==False: os.makedirs(tmp_dir)

#
depth_frames = sorted(glob.glob(os.path.join(scene_dir,'depth','*.png')))
print('find {} frames'.format(len(depth_frames)))

frame_count = 0
for depth_dir in depth_frames:
    frame_name = os.path.basename(depth_dir).split('.')[0]
    if DATASET=='scannet':
        frame_stamp = float(frame_name.split('-')[-1])
    else:
        raise NotImplementedError
    if frame_stamp % frame_gap != 0:
        continue
    print('integrating frame {}'.format(frame_name))

    # Load RGB-D and pose
    rgbdir = os.path.join(scene_dir,RGB_FOLDER,frame_name+RGB_POSFIX)
    pose_dir = os.path.join(scene_dir,'pose',frame_name+'.txt')
    if os.path.exists(pose_dir)==False:
        print('no pose file for frame {}. Stop the fusion.'.format(frame_name))
        break

    rgb = o3d.io.read_image(rgbdir)
    depth = o3d.io.read_image(depth_dir)
    T_wc = np.loadtxt(pose_dir)

    #
    rgbd = o3d.geometry.RGBDImage.create_from_color_and_depth(
        rgb,depth,depth_scale=DEPTH_SCALE,depth_trunc=DEPTH_SDF_TRUC,convert_rgb_to_intensity=False)
    # plt.subplot(1, 2, 1)
    # plt.title('RGB image')
    # plt.imshow(rgbd.color)
    # plt.subplot(1, 2, 2)
    # plt.title('depth image')
    # plt.imshow(rgbd.depth)
    # plt.show()
    pcd = o3d.geometry.PointCloud.create_from_rgbd_image( rgbd,o3d.camera.PinholeCameraIntrinsic(o3d.camera.PinholeCameraIntrinsicParameters.PrimeSenseDefault))
    # Flip it, otherwise the pointcloud will be upside down
    pcd.transform([[1, 0, 0, 0], [0, -1, 0, 0], [0, 0, -1, 0], [0, 0, 0, 1]])
    # o3d.visualization.draw_geometries([pcd])
    volume.integrate(rgbd, intrinsic, np.linalg.inv(T_wc))
```

semantic_mapping.py:The integrate semantic map function is tailored for semantic 3D scene reconstruction. Central to its operation is the management of an InstanceMap, which tracks and updates semantic instances within the scene. As it processes each frame, the function loads and filters detections, projects

these onto the 3D map, and either associates them with existing instances or creates new ones based on IoU thresholds and voxel resolutions. The function also refines and merges instances, updates global volume points, and integrates a dense TSDF volume for detailed scene reconstruction.

Here are some descriptions of some important functions in the module:
filter_overlap_detections: It calculates the Intersection over Union (IoU) for each pair of detections and normalizes their mask areas to derive scores, inversely related to their sizes. The core algorithm employs Non-Maximum Suppression (NMS) using these scores and the IoU matrix. NMS iteratively selects the detection with the highest score, discarding any overlapping detections exceeding a predefined IoU threshold.

```python
def filter_overlap_detections(detections:list(), min_iou:float=0.5):
    K = len(detections)
    if K<3: return detections
    ious = np.zeros((K,K),dtype=np.float32)
    volumes = np.zeros((K,),dtype=np.float32)

    # compute scores and iou
    for i, zi in enumerate(detections):
        uv_i = zi.mask # (H,W), bool, detection mask
        for j, zj in enumerate(detections):
            if j==i:ious[i,j]=0.0;continue
            uv_j = zj.mask # (H,W), bool, detection mask
            overlap = np.logical_and(uv_i,uv_j)
            ious[i,j] = np.sum(overlap)/(np.sum(uv_i)+np.sum(uv_j)-np.sum(overlap))
        volumes[i] = (np.sum(uv_i))

    volumes = volumes/np.max(volumes)
    scores = np.exp(-volumes)

    # nms
    pick_idxs = non_max_suppression(ious,scores,threshold=min_iou)
    if len(pick_idxs)<K:
        detections = [detections[k] for k in pick_idxs]
        print('!!!!!!!!!remove {} overlaped detections!!!!!!'.format(K-len(pick_idxs)))
    return detections
```

find_assignment: The function associates detected objects (from a list of detections) with instances in a 3D instance map. The function processes each detection and active instance, calculating the Intersection over Union (IoU) to establish a match based on spatial overlap. It creates an IoU matrix to evaluate the overlap between each detection and instance, and an assignment matrix to determine the best match, using a specified minimum IoU threshold. The algorithm also handles cases of multiple detections being assigned to a single instance, ensuring that each detection is uniquely matched to one instance, preferring the detection with the highest IoU. The function returns two elements: an array of matches indicating the instance matched with each detection (or -1 for unmatched detections), and a list of missed instances that weren't matched with any detection.

```python
def find_assignment(detections,instances,points_uv, min_iou=0.5,min_observe_points=200, verbose=False):
    '''
    Output:
        - assignment: (K,2), [k,j] in matched pair. If j=-1, the detection is not matched
    '''
    K = len(detections)
    M = instances.get_num_instances()
    if M==0:
        assignment = np.zeros((K,1),dtype=np.int32)
        return assignment, []

    # MIN_OBSERVE_POINTS = 200
    MIN_OBSERVE_NEGATIVE_POINTS = 2000

    # compute iou
    iou = np.zeros((K,M),dtype=np.float32)
    assignment = np.zeros((K,M),dtype=np.int32)
    # return assignment, []

    for k_,zk in enumerate(detections):
        uv_k = zk.mask # (H,W), bool, detection mask
        for j_ in range(M):
            l_j = instances.instance_map[str(j_)]
            p_instance = l_j.get_points()

            # if l_j.label != zk.label or p_instance.shape[0]==0: continue

            uv_j = points_uv[p_instance,1:3].astype(np.int32)
            observed = np.logical_and(uv_j[:,0]>=0,uv_j[:,1]>=0)
            if observed.sum() < min_observe_points: continue
            uv_j = uv_j[observed] # (|P_m|,2)

            uv_m = np.zeros(uv_k.shape,dtype=np.bool_)
            uv_m[uv_j[:,1],uv_j[:,0]] = True    # object mask
            if np.sum(uv_m)>0:
                overlap = np.logical_and(uv_k,uv_m)
                iou[k_,j_] = np.sum(overlap)/(np.sum(uv_m)) #+np.sum(uv_k)-np.sum(overlap))

    # find assignment
    assignment[np.arange(K),iou.argmax(1)] = 1

    instances_bin = assignment.sum(1) > 1
    if instances_bin.any(): # multiple detections assigned to one instance
        iou_col_max = iou.max(0)
        valid_col_max = np.abs(iou - np.tile(iou_col_max,(K,1))) < 1e-6 # non-maximum set to False
        assignment = assignment & valid_col_max

    valid_match = (iou > min_iou).astype(np.int32)
    assignment = assignment & valid_match
    # fuse instance confidence
    missed = []
    for j_ in range(M):
        instance_j = instances.instance_map[str(j_)]
        p_instance = instance_j.get_points()
        if p_instance.shape[0]>0:
            uv_j = points_uv[p_instance,1:3].astype(np.int32)
            observed = np.logical_and(uv_j[:,0]>=0,uv_j[:,1]>=0)
            if np.sum(observed)>MIN_OBSERVE_NEGATIVE_POINTS and assignment[:,j_].max()==0:
                instance_j.neg_observed +=1
                missed.append(j_)
```

association and integration: In this part of the script, the association and integration of detected objects into a 3D scene are executed. Initially, the find_assignment function associates each detection with corresponding instances in the 3D map using IoU metrics. Valid detections are then filtered for depth, transformed into point clouds, and integrated into the scene. New instances are created for unmatched detections, while existing instances are updated with new voxel data and labels for matched detections. This integration is tracked through projection_detections, and the script maintains a record of newly created or updated instances in recent_instances.

```
# Association
matches, missed = find_assignment(detections,instance_map,active_instances,min_iou=ASSIGNMENT_IOU,verbose=False)
t_3 = time.time()

# Integrate
count_matched = 0
projection_detections = []
for k, zk in enumerate(detections):
    valid_zk = label_predictor.is_valid_labels(zk.labels)
    if valid_zk==False: continue
    # or np.sum(zk.mask)<MIN_MASK_POINTS: continue

    # prepare detection mask
    depth_zk = filter_depth(depth_np, zk.mask, max_percentile=DEPTH_CLIP_MAX, min_percentile=0.1)
    depth_zk = o3d.geometry.Image(depth_zk)
    pcd_zk = generate_voxel_from_points(o3d.geometry.Image(rgb_np),depth_zk,VX_RESOLUTION,DEPTH_SCALE,T_wc,intrinsic)
    if len(pcd_zk.points)<MIN_MASK_POINTS: continue

    if matches[k] < -0.1:# new instance
        new_instance = fuse_detection.Instance(instance_map.max_instance_id,J=len(label_predictor.closet_names))
        new_instance.create_voxel_map(pcd_zk,VX_RESOLUTION)
        new_instance.prob_vector,flag = label_predictor.update_measurement(zk.labels,new_instance.prob_vector)
        if flag:
            new_instance.save_label_measurements(zk.labels)
            instance_map.insert_instance(new_instance)
            zk.assignment = 'new'
            recent_instances.append(str(new_instance.id))
    else: # matched instance
        matched_instance = instance_map.instance_map[str(matches[k])]
        matched_instance.prob_vector, flag = label_predictor.update_measurement(zk.labels,matched_instance.prob_vector)
        if flag:
            matched_instance.integrate_volume_debug(pcd_zk)
            matched_instance.save_label_measurements(zk.labels)
            count_matched +=1
            zk.assignment = 'matched'
            if str(matches[k]) not in recent_instances:
                recent_instances.append(str(matches[k]))

    projection_detections.append(zk)
```

5.2 Sketch-based animation Interface

The detailed description of codes is written below:

1. **MainController.cs**: This script controls the overall interactivity of this system.
2. **SystemUtils.cs**: This script contains helper functions to facilitate array operations, file loading, and geometry processing.
3. **MotionLine.cs**: This script contains motion line data structure and its functions. It stores two key information, the motion type and the 3D trajectory. It is constructed taking the 2D sketch points array as input and utilize the scene information to perform motion recognition and 3D trajectory reconstruction. Two key functions are shown in Figure.16.
4. **DrawHelper/DrawLine.cs**: This script contains line drawing helper functions that stores the mouse position and release points index to support sketch segmentation.
5. **DrawHelper/LineDrawer.cs**: This script controls the visualization of the 2D sketch line.

```
//// Recognize motion with dolloar recognizer
0 references
public void RecognizeMotion(DollarRecognizer recognizer){
    var resampled_2d = SystemUtils.Vector3To2(resampled_drawn_pts);
    var arc_length = SystemUtils.ComputeArcLength(resampled_drawn_pts);
    var arc_ratio = arc_length / Vector3.Distance(resampled_drawn_pts[0],resampled_drawn_pts[resampled_drawn_pts.Length-1]);
    if(arc_ratio<1.2){ //  straight line
        motion_name = "walk";
        Debug.Log(motion_name);
        return;
    }
    var res = recognizer.Recognize(resampled_2d);
    if(res.Match.Name.Contains("run")) motion_name = "run";
    else if(res.Match.Name.Contains("normal_jump")) motion_name = "normal_jump";
    else if(res.Match.Name.Contains("frontflip")) motion_name = "frontflip";
    else if(res.Match.Name.Contains("backflip")) motion_name = "backflip";
    if(motion_name.Contains("run")){
        resampled_drawn_pts = SystemUtils.ResamplePoints(drawn_pts,3.0f); // faster
    }
}
```

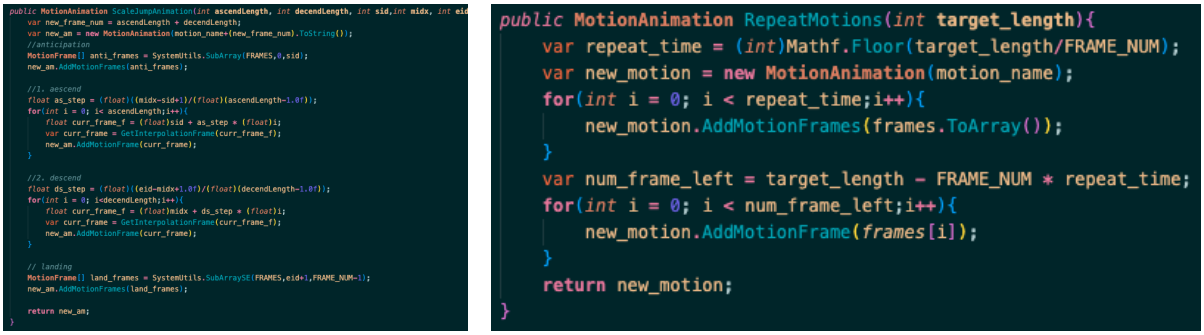
```
public void Generate3DLine(GameObject target_scene){
    var cam = Camera.main;
    /// For non-planar motion i.e. JUMP
    if(motion_name.Contains("jump")||motion_name.Contains("flip")){
        // only raycast the start and end point
        Vector3 spt, snorm, ept, enorm;
        spt = snorm = ept = enorm = Vector3.zero;
        GetHitPointAndNormal(resampled_drawn_pts[0],cam,target_scene,ref spt,ref snorm);
        GetHitPointAndNormal(resampled_drawn_pts[resampled_drawn_pts.Length-1],cam,target_scene,ref ept,ref enorm);
        trajectory_3d.Add(spt);
        for(int i = 1; i < resampled_drawn_pts.Length;i++){
            var ray = cam.ScreenPointToRay(resampled_drawn_pts[i]);
            Vector3 curve_pt = GetProjectedPointOnPlane(ray,spt,ept,spt+Vector3.up);
            trajectory_3d.Add(curve_pt);
        }
        trajectory_3d.Add(ept);
        return;
    }

    //// For motion lies on a plane
    var new_pts = SmoothGesturePoints(resampled_drawn_pts);
    for (int i = 0; i < new_pts.Length;i++){// loop points
        var ray = cam.ScreenPointToRay(new_pts[i]);
        RaycastHit Hit;
        if (Physics.Raycast(ray, out Hit))
        {
            for(int j = 0; j<target_scene.transform.childCount;j++){// loop scene objects
                var target = target_scene.transform.GetChild(j);
                if (Hit.transform == target.transform){
                    var hit_point = Hit.point;
                    trajectory_3d.Add(hit_point);
                    break;
                }
            }
        }
    }
    // modify motion for climb
    if(Mathf.Abs(trajectory_3d[0].y-trajectory_3d[trajectory_3d.Count-1].y)>0.1f){
        motion_name = "climb";
        Debug.Log(motion_name);
        Vector3 cpt, cnorm;
        cpt = cnorm = Vector3.zero;
        GetHitPointAndNormal(resampled_drawn_pts[0],cam,target_scene,ref cpt,ref cnorm);
        climb_norm = cnorm;
        for(int i = 0; i < trajectory_3d.Count;i++){
            trajectory_3d[i] += cnorm * 0.15f;
            trajectory_3d[i] -= new Vector3(0.0f,0.2f,0.0f);
        }
    }
}
```

Figure 16: Key functions for MotionLine.cs Recognize motion type(Top). Generate 3D trajectory(Bottom)

6. **DollarReccognizer/DollarRecognizer.cs**: This script contains the implementation of the $1 recognizer[10]. We use an existing implementation with open licenses and add small modifications to let it support XML gesture data reading.

7. **AnimationEditor/MotionAnimation.cs**: This script contains the definition of two animation classes: MotionFrame and MotionAnimation. The MotionFrame class stores the rotations of each joints at one single frame. The MotionAnimation class is consisted of multiple motion frames. It also contains helper function to perform the frame interpolation as shown in Figure. 17.



```
public MotionAnimation RepeatMotions(int target_length){
    var repeat_time = (int)Mathf.Floor(target_length/FRAME_NUM);
    var new_motion = new MotionAnimation(motion_name);
    for(int i = 0; i < repeat_time;i++){
        new_motion.AddMotionFrames(frames.ToArray());
    }
    var num_frame_left = target_length - FRAME_NUM * repeat_time;
    for(int i = 0; i < num_frame_left;i++){
        new_motion.AddMotionFrame(frames[i]);
    }
    return new_motion;
}
```

Figure 17: Key functions for MotionAnimation.cs Jump frame interpolation(Left). Motion repeat(Right)

8. **AnimationEditor/FBXAnimationLoader.cs**: This script contains helper function to load motion preset from fbx file.

9. **AnimationEditor/AvatarControl.cs**: This script is attached to the avatar object to control the movement between frames. The MainController will assign combined motion animation(including trajectory and motions) to this AvatarController to enable the avatar to move at each frame. The deatils are shown in Figure. 18.

```
if(is_playing){
    MotionFrame curr_frame = avatar_animaiton.GetFrame((int)frame_timer);
    MoveJointAtFrame(root_joint, curr_frame);
    transform.position = trajectory_3d[(int)frame_timer];
    transform.rotation = body_rots[(int)frame_timer];
    frame_timer += Time.fixedDeltaTime * motion_speed;
    if(frame_timer >= trajectory_3d.Count-1 || frame_timer >= avatar_animaiton.FRAME_NUM -1 ){
        frame_timer = 0.0f;
        is_playing = false;
    }
}
```

```
2 references
public void MoveJointAtFrame(Transform root, MotionFrame mf){
    root.localRotation = mf.GetJointRotation(root.name);
    for(int i = 0; i < root.childCount;i++){
        MoveJointAtFrame(root.GetChild(i),mf);
    }
}
```

Figure 18: Code in AvatarControl.cs to control avatar motion at each frame(top) and rotate joints recursively(bottom).

10. **AnimationEditor/AnimationModifier.cs**: This script contains functions to composite multiple motion sketches into one single animation. It compute the final frame numbers, avatar look at directions and frame interpolations. The details are shown in Figure. 19

5.3 Code

The code of this project can be viewed from Github repository

```
public MotionAnimation ModifyMotion(MotionLine motion_line, int ase_frame, int dse_frame){
    // one frame at a point
    var new_motion = new MotionAnimation(motion_line.MOTION+"_modified");
    if(motion_line.MOTION.Contains("normal_jump")) {// jump
        new_motion = motion_dict[motion_line.MOTION].ScaleJumpAnimation(ase_frame, dse_frame, 23,31, 42);
    }else if(motion_line.MOTION=="frontflip"){
        new_motion = motion_dict[motion_line.MOTION].ScaleJumpAnimation(ase_frame, dse_frame, 0,16, 32);
    }else if(motion_line.MOTION=="backflip"){
        new_motion = motion_dict[motion_line.MOTION].ScaleJumpAnimation(ase_frame, dse_frame, 34,45,59);
    }
    else {// walk or running or climb
        new_motion = motion_dict[motion_line.MOTION].RepeatMotions(motion_line.LENGTH);
    }
    return new_motion;
}
```

```
void ComputeBodyRotation(int curr_line_start, MotionLine motion_line){
    for(int i = curr_line_start; i < curr_trajectory.Count;i++){
        // get moving direction
        var move_dir = Vector3.right;
        if(i==0) {
            move_dir = curr_trajectory[i+1] - curr_trajectory[i];
        }
        else {
            move_dir = curr_trajectory[i] - curr_trajectory[i-1];
        }
        if(motion_line.MOTION.Contains("jump")||motion_line.MOTION.Contains("frontflip")){
            move_dir = curr_trajectory[curr_trajectory.Count-1] - curr_trajectory[curr_line_start];
        }else if(motion_line.MOTION=="backflip"){
            move_dir = curr_trajectory[curr_line_start] - curr_trajectory[curr_trajectory.Count-1];
        }else if(motion_line.MOTION.Contains("climb")){
            move_dir = motion_line.CLIMBN;
        }
        move_dir.y = 0.0f; // do not change the up direction value
        var avatar_quat = Quaternion.LookRotation(-move_dir.normalized);
        curr_quats.Add(avatar_quat);
    }
}
```

Figure 19: Code in AnimationModifier.cs to combine animation clips. The base motion clips are processed based on motion types(top). The body rotations are also computed follows the motion and trajectory information(bottom).

6 Limitations and Future work

6.1 Semantic integration

The semantic labels in this project are produced by Grounded-SAM.[5][7][13] The automatic labeling module includes GroundingDINO for object detection, SAM (Segment Anything Model) for segmentation, and Recognize Anything Model (RAM) for tag generation. It processes images from ScanNet, where each image undergoes tag generation with RAM, object detection with GroundingDINO, and segmentation with SAM. The script validates and augments generated tags, filters detection outputs using thresholds, and segments each detected object, creating precise masks. It saves and visualizes the results, overlaying masks and boxes on original images for reference.

However, the semantic labels are not fully utilized in the project, which at the beginning we presumed we could use these semantic labels for more augmentation of the 3D animation. In the future, maybe we could use these semantic labels generate scene graph(e.g. modified from SceneGraphFuson[11]) to help animators to create animations just by inputting high level descriptions(e.g. ”let the charactor jump from the desk A to chair B”).

6.2 Improve scene geometry quality

There are sill some misalignment between estimated scene geometry and point cloud, as shown in Figure.20 This is due to the voxelization of the pointcloud for simplified generation of the object mesh. In the future, we may apply a better algorithm to generate high quality object meshes of few faces from the pointclouds.

Figure 20: misalignment between estimated scene geometry and point cloud

6.3 More realistic physics

The current system does not include any physical simulation because the scene geometry is still in a rough shape, which could lead to errors in simulation due to inconsistent surfaces and slopes. In the future, we aim to improve the quality of our generated mesh and include physics-based character animation to achieve more realistic effects.

6.4 User study

In the current evaluation, we only compared the time used for creating similar animation with a commercial software blender. To further prove the usability of our system, we could perform user study with questionaires such as NASA Task Load Index , which is widely used in examine the usability of interactive systems [4].

7 Conclusion

In conclusion, we have developed a interactive animation system with 3D scene understanding to support easy-creation of avatar animation. The system includes 3D scene understanding to process raw scene data for animation creation and adopts a sketch-based animation interface to improve user experience. The current system can be used for creating simple animation that consists of 6 different motions. In the future, we will improve this system with better scene geometry estimation algorithm, physical-based character animation, and semantic integration. The system will also be evaluated with real target users to evaluate its usability and performance in comparison with other commercial software.

8 Work Distribution

8.1 ZHENG Kexin

1. Design and develop sketch-based animation creation pipeline
2. Design user interface and interactivity
3. Develop final interactive application

8.2 HUANG Haoming

1. Dataset preparation and dense mapping pipeline
2. Instance segmentation
3. Semantic prediction exploration

References

[1] Dmitry Chetverikov. A simple and efficient algorithm for detection of high curvature points in planar curves. In *International Conference on Computer Analysis of Images and Patterns*, pages 746–753. Springer, 2003.

[2] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[3] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

[4] Sandra G Hart. Nasa-task load index (nasa-tlx); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 50, pages 904–908. Sage publications Sage CA: Los Angeles, CA, 2006.

[5] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[6] Vojtěch Krs, Ersin Yumer, Nathan Carr, Bedrich Benes, and Radomír Měch. Skippy: Single view 3d curve interactive modeling. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017.

[7] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[8] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*, pages 347–353. 1998.

[9] Matthew Thorne, David Burke, and Michiel Van De Panne. Motion doodles: an interface for sketching character motion. *ACM Transactions on Graphics (ToG)*, 23(3):424–431, 2004.

[10] Jacob O Wobbrock, Andrew D Wilson, and Yang Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 159–168, 2007.

[11] Shun-Cheng Wu, Johanna Wald, Keisuke Tateno, Nassir Navab, and Federico Tombari. Scenegraphfusion: Incremental 3d scene graph prediction from rgb-d sequences. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7515–7525, 2021.

[12] Hui Ye, Kin Chung Kwan, Wanchao Su, and Hongbo Fu. Aranimator: In-situ character animation in mobile ar with user-defined motion gestures. *ACM Transactions on Graphics (TOG)*, 39(4):83–1, 2020.

[13] Youcai Zhang, Xinyu Huang, Jinyu Ma, Zhaoyang Li, Zhaochuan Luo, Yanchun Xie, Yuzhuo Qin, Tong Luo, Yaqian Li, Shilong Liu, et al. Recognize anything: A strong image tagging model. *arXiv preprint arXiv:2306.03514*, 2023.