This article will show you how to build a mixed-reality game using Intel® RealSenseTM Camera (rear), Intel® RealSenseTM Software Development Kit, Unity 3dTM Game Engine, and changing terrain (sand). Game creators are always looking for new ways to improve the user experience and get players into the game. From avatars and deep game options to terrain editors and custom mini-games, developers are constantly working to give users a sense of virtual presence.

Intel® RealSenseTM technology brings many exciting new ways for players to interact with their computers. TANKED! is an augmented reality sandbox arcade game that uses this technology in existing multiplayer battles. Design Mill built a custom sandbox and then used Intel RealSense technology and an image projector to support the user's hands to manipulate the terrain of the map, creating a virtual environment dynamically through real-world sand.

TANKED! is an arcade-style one-on-one tank battle video game. Players first need to use real sand to build a virtual environment. After modifying the terrain, the tanks provided for each player will fall into the environment and the game begins. The player manipulates the tank to travel on the dynamically generated terrain, using the slope to launch an attack while avoiding other players' projectiles.

Tanks can be driven into the valley to seek cover, or climb the hill to adjust the range of the cannon, while also gaining the props deployed between the strategies during the game. State acceleration, faster tank speeds, and cannon firing rates can make a big difference in combat. The player who first fires enough gunfire at the opponent will win the game.

However, there is also a level: sweeping mini games. The system reminds the player that there is a mine and the tank stops fighting. The deposits are randomly distributed, with the Intel® RealSenseTM camera to determine if the depth of the sand is suitable for burying the deposit. Before the time spent, each player must dig the sand to find the minerals and make them inactive.

After discovering the mineral deposit, the player needs to press the operation button to make the mineral deposit inactive. If the player is unable to make the mineral deposits in the area inactive, it will cause a lot of damage to his tank.

Intel RealSense cameras and image projectors are installed in a rectangular shag designed for two players. Each player has a joystick that controls the tank, an action button that fires a cannonball or dismantles the mine, and a start button.

The Intel RealSense camera will align with the inside of the sandbox, ignoring everything on the wall and beyond. Then we project the image of the created virtual terrain onto the sand.

Use the Intel® RealSenseTM Software Development Kit to capture depth data from sand, enabling control of virtual terrain in a unique tactile manner. With the Intel RealSense Software Development Kit, we can adjust sensors and projectors, read sand depth and eliminate and replace invalid depths.

The changing terrain will adjust the path of the projectile based on the depth data in the Intel RealSense Software Development Kit. If the tank is on the mountain, the projectile will be fired into the air at a greater angle. Players can then build hills for protection. Therefore, each gameplay is unique because the dynamic terrain created by the player using sand varies. Players must be aware of their vertical position in natural space, the distance to the enemy, and whether their launch distance can be shot.

With the Unity* 3D game engine, the Intel RealSense camera enables dynamic terrain creation. Since the Intel RealSense camera is mounted directly above the Shag, we used COORDINATE_SYSTEM_REAR_OPENCV SETTING for the sensor and captured the STREAM_TYPE_DEPTH data stream. For each frame, Unity's Update method will call the code that will instruct the Intel RealSense camera to transfer the depth data for each point on the sand to Unity.

To ensure high performance, depth data within an acceptable range is evenly distributed over a specific number of frames within Unity. After averaging, the depth will be reduced, rotated, compensated, and cropped as needed. These depths are then expanded to fill the array with the size of the Unity terrain heightmap resolution. The set height is then pushed to the TerrainData.SetHeights method in Unity. Then, the splatmap textures we customized in Unity will be updated to reflect these new heights.

Finally, dynamically generated terrain with textures and plants will be mapped to Shag; this will make each player feel like driving a tank in the world they created. Although this process sounds long, it doesn't actually take too long and can be done seamlessly in the game stream.

In the mine sweeping mini-game, the terrain was re-adjusted after the player moved the sand to find the mine. This demonstrates the dynamic environment that can be created using Intel RealSense cameras. Players can experience hidden worlds and buried secrets in their own created environment.

```
/// <summary>
/// Gathers depth data from the sensor, adding it to the points you are accumulating for averaging purposes
/// If the number of frames accumulated has reached the number you want to average, the terrain itself if refreshed.
/// </summary>
private void updateTerrain()
{
   if (pp == null || !autoUpdate)
      return;

   pxcmStatus sts = pp.AcquireFrame (false, 0);
   if (sts == pxcmStatus.PXCM_STATUS_NO_ERROR)
{
   PXCMCapture.Sample sample = pp.QuerySample();

   If (sample.depth != null)
{
   cumulativeFramesCount = GetRGB32PixelsDepth2 (sample.depth);
   if (cumulativeFramesCount >= averagePasses)
      cumulativeFramesCount = RefreshData2();
}

pp.ReleaseFrame();
}
else
   pp.ReleaseFrame();
}


/// <summary>
/// Populates the pointsToAverage list with the data from the sensor
/// </summary>
/// <returns> The number of frames accumulated
/// <param name=”image”>Depth sample from the sensor</param>
private int GetRBG32PixelsDepth2 (PXCMImage image)
{
   PCXMImage.ImageData cdata;
   UInt16[] dpixels = null;
```

```
    if (image.AcquireAccess(PXCMImage.Access.ACCESS_READ,
PXCMImage.PixelFormat.PIXEL_FORMAT_DEPTH, out cdata) >=
pxcmStatus.PXCM_STATUS_NO_ERROR)
{
    Int32 dpitch = cdata.pitches[0] / sizeof(Int16);
    Int32 dwidth = (Int32) image.info.width;
    Int32 dheight = (Int32) image.info.height;
    dpixels = cdata.ToShortArray(0, dpitch * dheight);
    image.ReleaseAccess(cdata);
    cumulativeFramesCount += 1;

    float testValue;
    foreach (PointForAveraging point in pointsToAverage)
    {
        //initialize based on the coordinates
        testValue = dpixels[point.heightCoord * dpitch + point.widthCoord];

        if (testValue != depthLowConfidenceValue
        && testValue >= realSenseSettings.MinDepthValue
        && testValue <= realSenseSettings.MaxDepthValue)
        {
            point.sumZValues += testValue;
            point.countZValues += 1;
}
}
}

return cumulativeFramesCount;
}
```

The location and placement of Intel RealSense cameras and projectors is critical to the success of the game. Intel RealSense cameras must be installed away from Shag to get the wall. We created a calibration utility to help the player select the area of the sandbox by setting the x and y axes and width of the display area. These coordinates indicate whether the sensor needs to be mapped and whether the depth point needs to be rotated 180 degrees before it is displayed in the sand. This utility calculates the height of the area to display based on the selected width and aspect ratio of the Intel RealSense camera.

The calibration settings will determine the depth points that will be used in the actual game. The depth points are collected according to each frame in Unity 3D. To ensure high game performance, these depth point data are evenly distributed over a specific number of frames. Data with less confidence in the depth sensor and depth points outside the specified range are ignored. Any ignored depth values are filled with the nearest effective pixel depth value.

Unity 3D terrain is a square object, we need to fill in a rectangular sandbox. Therefore, before we update the terrain height, the effective depth points are measured to match the terrain resolution by using the effective width and depth pixels: the number of heightmap resolutions. We create a set of heightmap resolutions using the measured values and push them to the TerrainData.SetHeights() method. The terrain texture and trees are then updated based on these new heights so that the player can see the updated terrain.

Sometimes, due to the poor quality of the depth data, holes will appear on the terrain, which will cause the tank to fall into the map. We solved this problem by reading the pixels around the

moving tank instead of a single point. In addition, we created a script for the tank layer so that the tank can move smoothly on the plane.

Create a dynamic environment creation framework with the Intel RealSense Software Development Kit to create a new digital environment opportunity. This applies not only to entertainment games, but also to an educational interactive experience. Whether the gameplay is primarily for entertainment or education, the virtual sand provided by Intel RealSense technology provides a new entertainment and learning experience.

The development of TANKED! is a fun and educational process and we hope to continue to improve the game in the future. We will continue to improve the gameplay and optimize it to provide better frame rates and seamless control for a better user experience.

In addition to developing TANKED!, we have also developed an interpretive projector system developer kit (called Torch), which is currently available soon. This enables the developer community to create their own interactive sandbox and desktop experience. Learn more about the Torch platform: www.ExperienceTorch.com.

The Intel RealSense Software Development Kit documentation, Samples Viewer, and code samples are essential tools for developing TANKED! The Raw Streams and Capture Viewer examples describe the color, depth, and infrared data streams of Intel RealSense cameras from different dimensions and frame rates. These examples show us what the camera can capture so we can take advantage of it in the game. The Unity and C# code examples provided in the Software Development Kit provide a good foundation for our development work. The Intel Developer Zone, the personal support of the Intel RealSense Technology team, the powerful community and support of the Unity game engine provide additional tools and resources for the implementation of the game.

About Design Mill (www.designmillinc.com)
Design Mill, Inc. is a strategic systems integrator that provides leading-edge solutions for leading companies and the Department of Defense to transform advanced interactive hardware and software through innovative design, development, and process integration. Design Mill focuses on next-generation virtual reality and augmented reality solutions to help customers deliver revolutionary user interaction methods and untapped development strategies for a win-win situation.