Version 1.0.0

Project Dawn | Adaptive Split Screen | Tool

Table of Contents

Overview

This package contains fast lightweight solution for split screen. It is developed with DOTS in mind, as result it takes advantage of Unity latest technology stack like SIMD mathematics, Jobs and Burst compiler.

Manual

This is novel solution of split screen based on voronoi diagrams. One can think of it as extension of traditional dynamic split screen for up to 4 players.

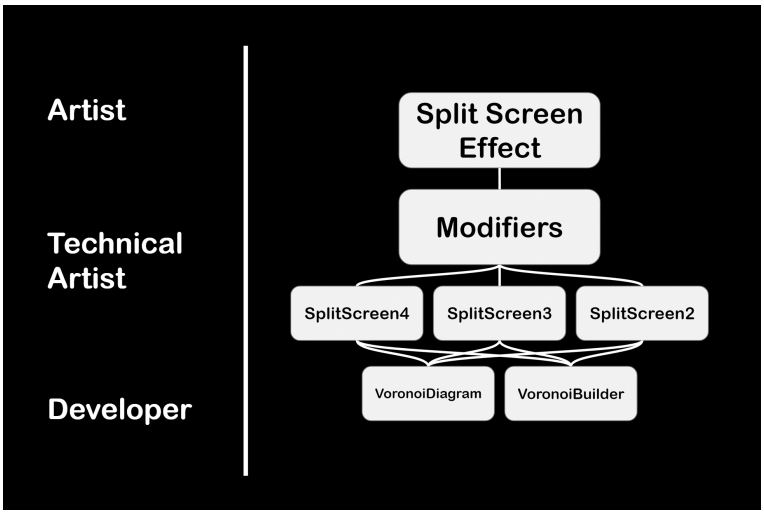
Key Features:

- **Fairness** - option for each player to have similar screen region area.
- **Centered** - option for each player to be centered in their screen region.
- **Direction Indicated** - each split indicates direction to other player.
- **Fusible** - players screen regions can be merged once they are close enough.

> **Note:** Currently 4 sites balancing comes with the cost of continuity. Check limitations section for more details.

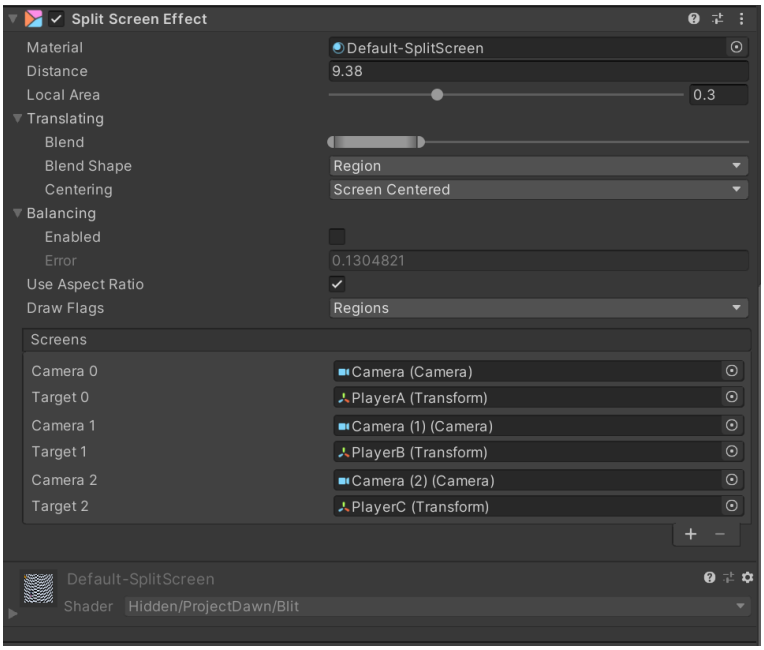
The package is designed with scalability in mind. Thus it is composed from multiple levels of API layers. Here is quick overview of it:

1. `SplitScreenEffect` - the main MonoBehaviour component that applies split screen effect on camera.
2. `Modifier` - custom MonoBehaviour components for modifying logic of `SplitScreenEffect`.
3. `SplitScreen4`, `SplitScreen3` and `SplitScreen2` - structs for generating split screen effect.
4. `VoronoiBuilder`, `VoronoiDiagram` - structs for generating voronoi diagrams from sites.



Artist Path

For artists there is zero code path using `SplitScreenEffect` MonoBehaviour component. For quick creation use menu item `GameObject/Rendering/Split Screen`.



Material

Material used for drawing each screen region. Default material can be set in two ways:

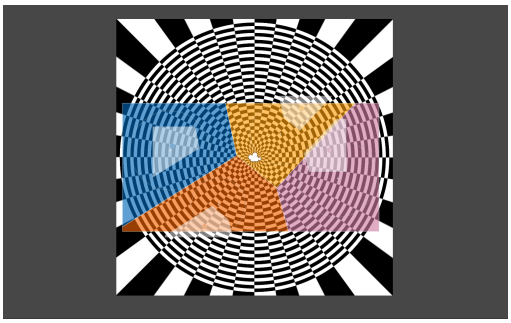
- Setting material to none in inspector view.
- In scripting setting material to `SplitScreenResources.DefaultMaterial`.

Distance

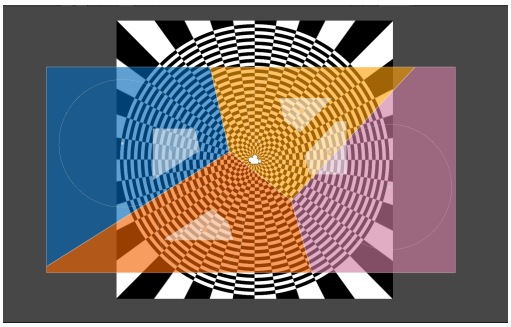
Distance from the each screen target in world space.

BoundsRadius

Radius that is used for construcing bounds.



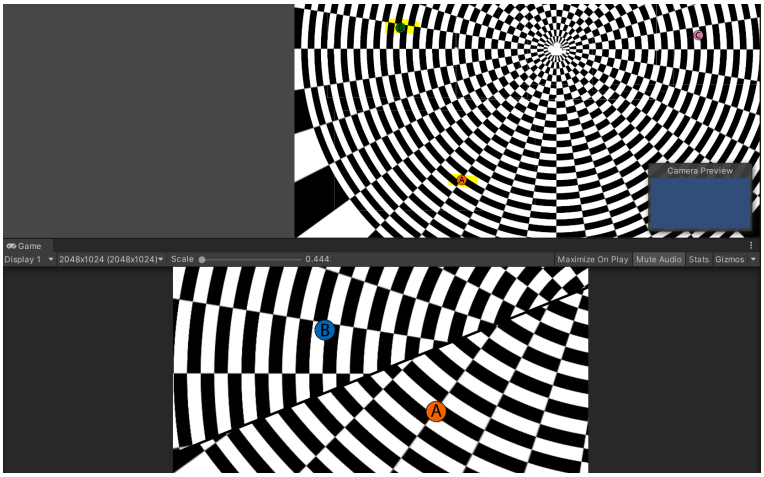
Bound radius smaller number. Colored polygons represents player screen region.



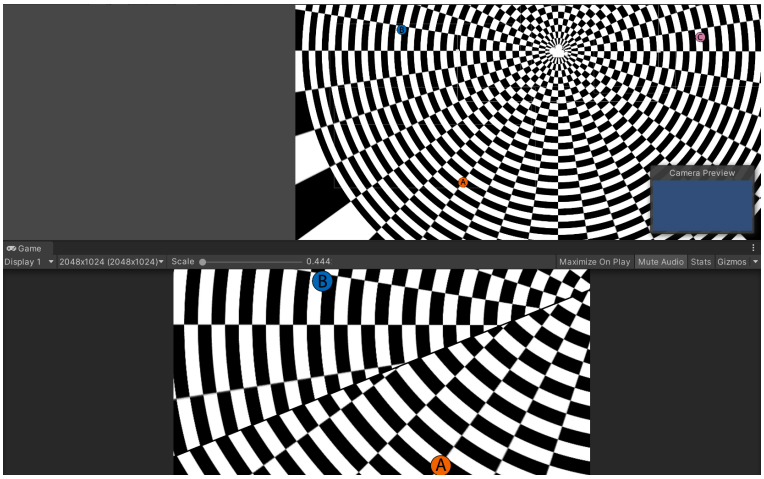
Bound radius bigger number. Colored polygons represents player screen region.

Blend

Controls range from which player's views starts and stops blending.



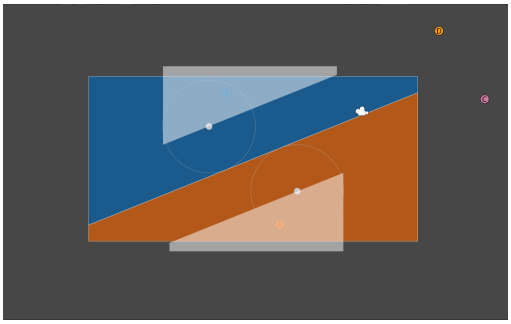
When blend value is low player's views are not blending.



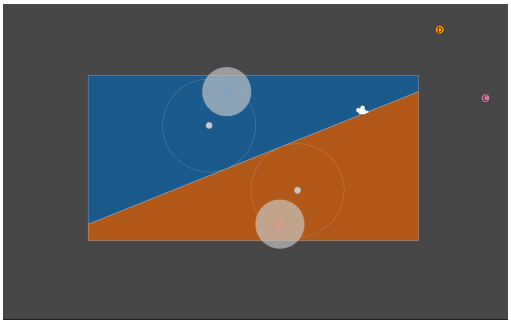
when blend value is quite high the player's views start blending.

Blend Shape

Shape used for calculating distance between players.



Region is more accurate, but less smooth.

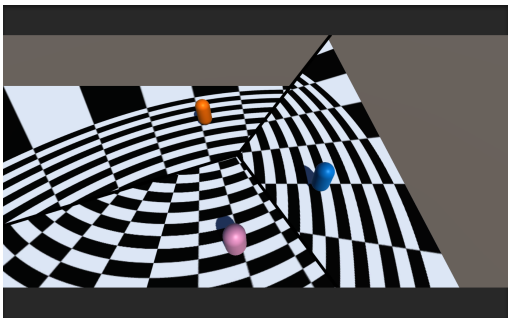


Circle is less accurate, but more smooth.

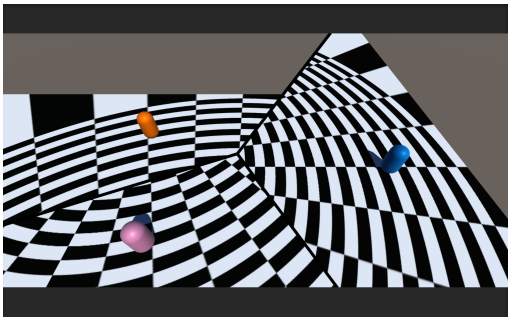
Centering

Controls how players are centered within it's screen region.

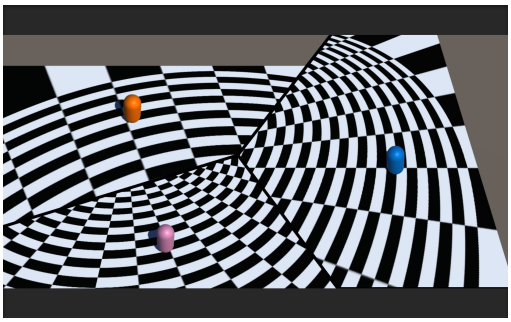
> **Note:** Centering will result in erratic movement as player is offset from his real position. Balancing usually ease out this side effect.



No centering.



Screen centred.



Players centred. In perspective camera view it keeps focus on player instead of screen's center.

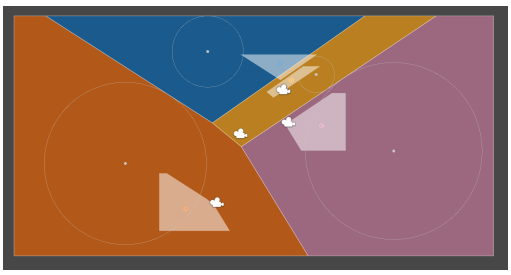
> **Note:** Currently player centering results in same cases with black border, it is known limitation.

Balancing

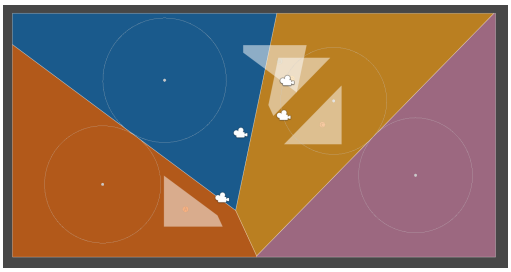
Balancing option archieves two points. Firstly, reduces erratic movement using centering. Secondly, balances screen regions near equal area.

For 3 players optimization happens on translating and scaling all sites using Levenberg Marquardt.

For 4 players there is no stable algorithm, it uses combination of two algorithms. Translating with Levenberg Marquardt and Floyd's Relaxation.



Unbalanced.



Balanced.

Technical Artist Path

Scripting

Here is small sample that shows, how to add screen data into `SplitScreenEffect` using scripting.

```
splitScreen.Clear();
for (int screenIndex = 0; screenIndex < screenCount; ++screenIndex)
    splitScreen.AddScreen(cameras[screenIndex], targets[screenIndex]);
```

Modifiers

For minimal modifications to `SplitScreenEffect` there is a special design called `Modifier`. Which essentially is a collection of interfaces that get automatically called at specific places in split screen.

All you need to do is create `MonoBehaviour` that implements one of these interfaces and it to same game object as `SplitScreenEffect`:

```
public interface ISplitScreenTargetPosition
{
    float3 OnSplitScreenTargetPosition(int screenIndex, float3 positionWS);
}

public interface ISplitScreenCommandBuffer
{
    void OnSplitScreenCommandBuffer(CommandBuffer commandBuffer, in ScreenRegions screenRegions);
}

public interface ISplitScreenBalancing
{
    void OnSplitScreenBalancing(VoronoiBuilder voronoiBuilder, VoronoiDiagram voronoiDiagram,
NativeArray<float2> sites, in Balancing balancing);
}
```

Package already provides few common modifiers like:

- `DrawSplitsModifier` - for drawing split lines.
- `SmoothModifier` - for more smooth motion of voronoi.

Developer Path

For those who interested in extensive changes to the package or even implementing your own version of `SplitScreenEffect` MonoBehaviour component. It is recommended to check `SplitScreen4`, `SplitScreen3` and `SplitScreen2` that allows generating screen regions.

```
public Transform TargetA;
public Transform TargetB;
public Transform TargetC;

void OnDrawGizmos()
{
    if (TargetA == null || TargetB == null || TargetC == null)
        return;

    // Create split screen
    var splitScreen = new SplitScreen3(Allocator.TempJob);
    splitScreen.Reset(TargetA.transform.position, TargetB.transform.position, TargetC.transform.position, 2,
1, 0.2f);

    // Creates screen regions
    var screenRegions = new ScreenRegions(Allocator.TempJob);
    splitScreen.CreateScreens(Translating.Default, ref screenRegions);

    // Draw screen regions
    for (int screenRegionIndex = 0; screenRegionIndex < screenRegions.Length; ++screenRegionIndex)
    {
        float3 cameraPositionWS = screenRegions.Regions[screenRegionIndex].Position;
        Gizmos.DrawConvexPolygon(screenRegions.GetRegionPoints(screenRegionIndex), 1, cameraPositionWS.xy,
PlayerColor.Players[screenRegionIndex]);
    }

    // Cleanup
    splitScreen.Dispose();
```

```
        screenRegions.Dispose();
    }
```

Small sample of constructing split regions.

Split screen at its core uses voronoi diagrams to generate screen regions.

```
public Transform[] Targets;
public Rect Rect;

void OnDrawGizmos()
{
    if (Targets == null)
        return;

    using (var builder = new VoronoiBuilder(Allocator.Temp))
    {
        // Gather sites
        var sites = new NativeList<float2>(Targets.Length, Allocator.Temp);
        foreach (var target in Targets)
        {
            if (target)
                sites.Add(((float3) target.transform.position).xy);
        }
        builder.SetSites(sites);

        // Create voronoi
        var voronoiDiagram = new VoronoiDiagram(Allocator.Temp);
        builder.Construct(ref voronoiDiagram, Rect);

        // Draw voronoi
        foreach (var region in voronoiDiagram.Regions)
            Gizmos.DrawConvexPolygon(voronoiDiagram.GetRegionPoints(region), 1, 0,
PlayerColor.Players[region.Index]);

        // Cleanup
        sites.Dispose();
        voronoiDiagram.Dispose();
    }
}
```

Small sample of constructing voronoi diagram and drawing it.

Here are few links that will help understand core funcionality of split screen:

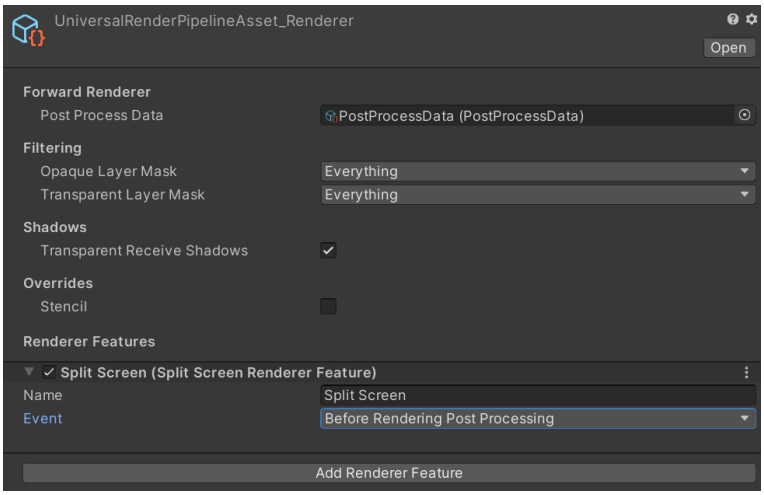
- Juicing Your Cameras With Math 2016 GDC
- Fair Voronoi Split-Screen

Builtin Render Pipeline

No setup is needed. It works by automatically passing command buffer into main camera using `Camera.AddCommandBuffer`.

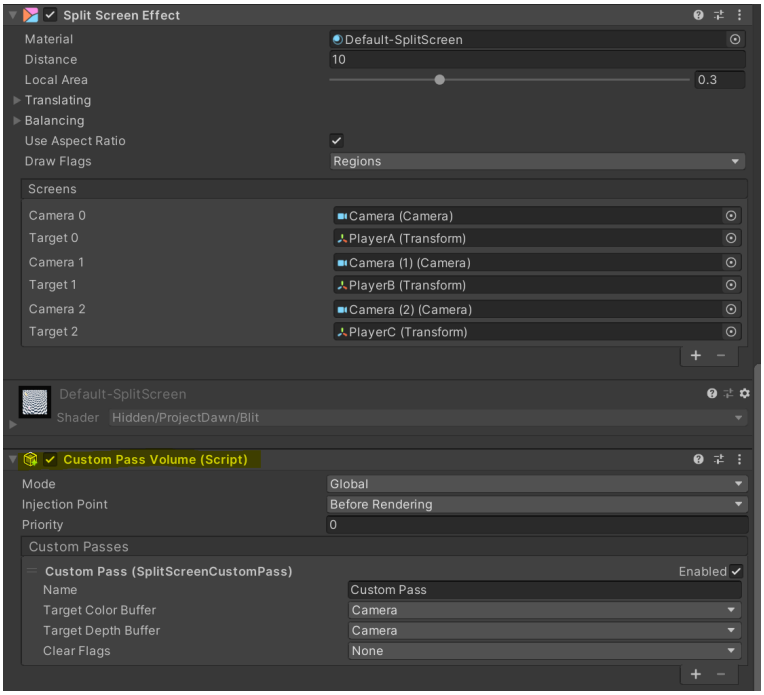
Universal Render Pipeline

For `SplitScreenEffect` to work you need to add `SplitScreenRendererFeature` into your renderer. Also make sure that your camera has that renderer set as target. Check URP page for more information about RendererFeatures.



High-Defenition Render Pipeline

For `SplitScreenEffect` to work you need to add `CustomPassVolume` component inside same game object with custom pass `SplitScreenCustomPass`.



> **Note:** Only works from 10.5 version, because there is a bug with post processing ordering 7.7.1. There is workaround if you are interested in it please contact me.

Custom Render Pipeline

For `SplitScreenEffect` to work you need to execute command buffer `SplitScreenEffect.GetCommandBuffer()` at post processing event.

Additional API

There are lots of other quite usefull APIs that can be used outside of this package.

- `ConvexPolygon` - operations with convex polygon (etc. calculting area, getting centroid...)
- `MaxInscribedCircle` - finding max inscribed circle in convex shapes.
- `Line`, `StandardLine` - operations with the lines (etc. intersection point, distance...)
- `Circle` - operations with circle.

- `DisabledInInspector`, `MaxValue`, `MinMaxRange`, `Reload` - property attributes.
- `Blend` - operations for blending line/triangle and quad.

Limitations

- Works only up to 4 split screens. This is mainly for two reasons. First, unity is not that great when it comes cpu cost of multiple cameras. Second, I felt like it is more rare case to have more than 4 split screens in co-op. This is a subject to change if there would be a need.
- Screen balancing for 3 split screens works quite perfect. However this is not the case for 4 split screens as it requires more complex solution, which tends to erratic movement of splits. This might be improved in the future.
- Player centering currently produces black borders in some cases. This is in subject to change in the future.
- Post processing with `SplitScreenEffect` might not be that straight forward depending on effect. As it combines screen regions and depth information gets lost there.

Dependencies

- Tested with Unity 2019.4
- Package com.unity.mathematics@1.2
- Package com.unity.collections@0.9
- Package com.unity.burst@1.4

Support

If you have questions, bugs or feature requests, use Discord.