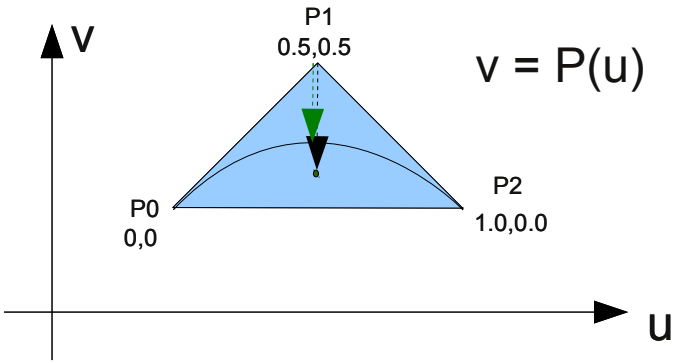
Simple Quadratic, n=2



$$\mathbf{p}(u) = \Sigma_{i=0}^{n} \mathbf{p}_i N_{i,D}(u)$$

where D=3, n=2

Knot vector=[0 0 0 1 1 1]

v = P(u)

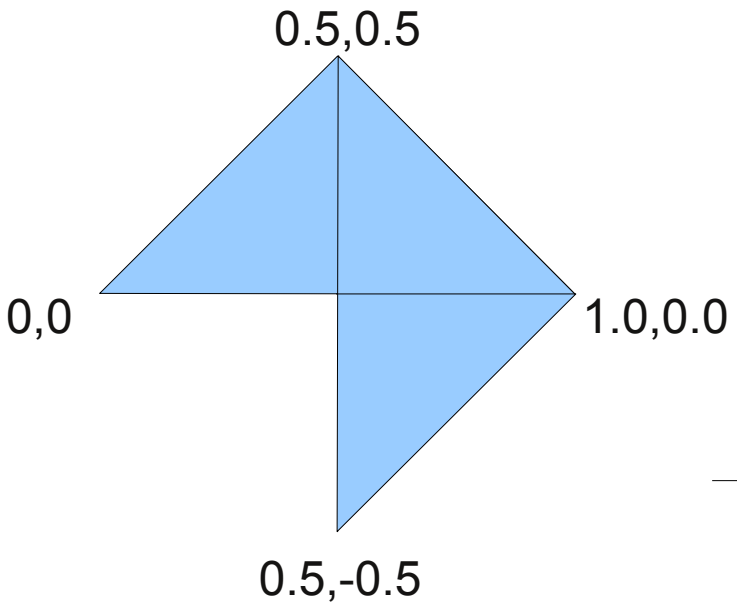
If interior is required, we change sign of P1v
→ P1(0.5,-0.5), wont affect the algorithm

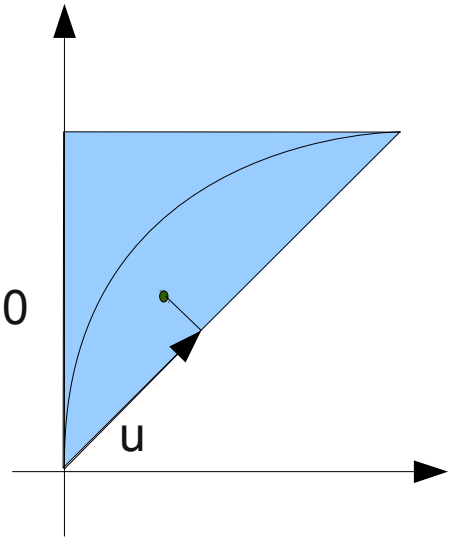
Let A=(u,v)=(u,P(u))

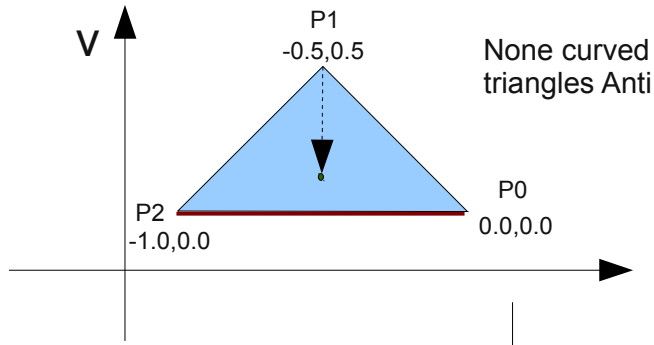
Let B=(u,abs(v)) → current pixel fragment

```
if(( v > 0  and ||P1B|| > ||P1A||)
   or ( v < 0 and ||P1B|| < ||P1A||))
        color = c;
        c.alpha = func(P1B,P1A, sign(v))
else
      kill
```

Generalization, n > 2







None curved boundary triangles Anti aliasing

Since P0P2 is a boundary edge
→ the boundary edge case can be defined by signs different of the P2u.

Hence

```
if (u < 0)
   set color = c;
   Let P1A = (abs(u), 0)
   P1B = (abs(u), v)
   color.alpha = func(P1B,P1A,0)
```

Note: with this addition our shader will be able to handle the boundary triangles and Interior triangles.

No need for shader switching.