Ray × Bèzier surface intersection

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

Bicubic Bèzier patch

$$P_{ij} = \left[x_{ij}, y_{ij}, z_{ij} \right]$$

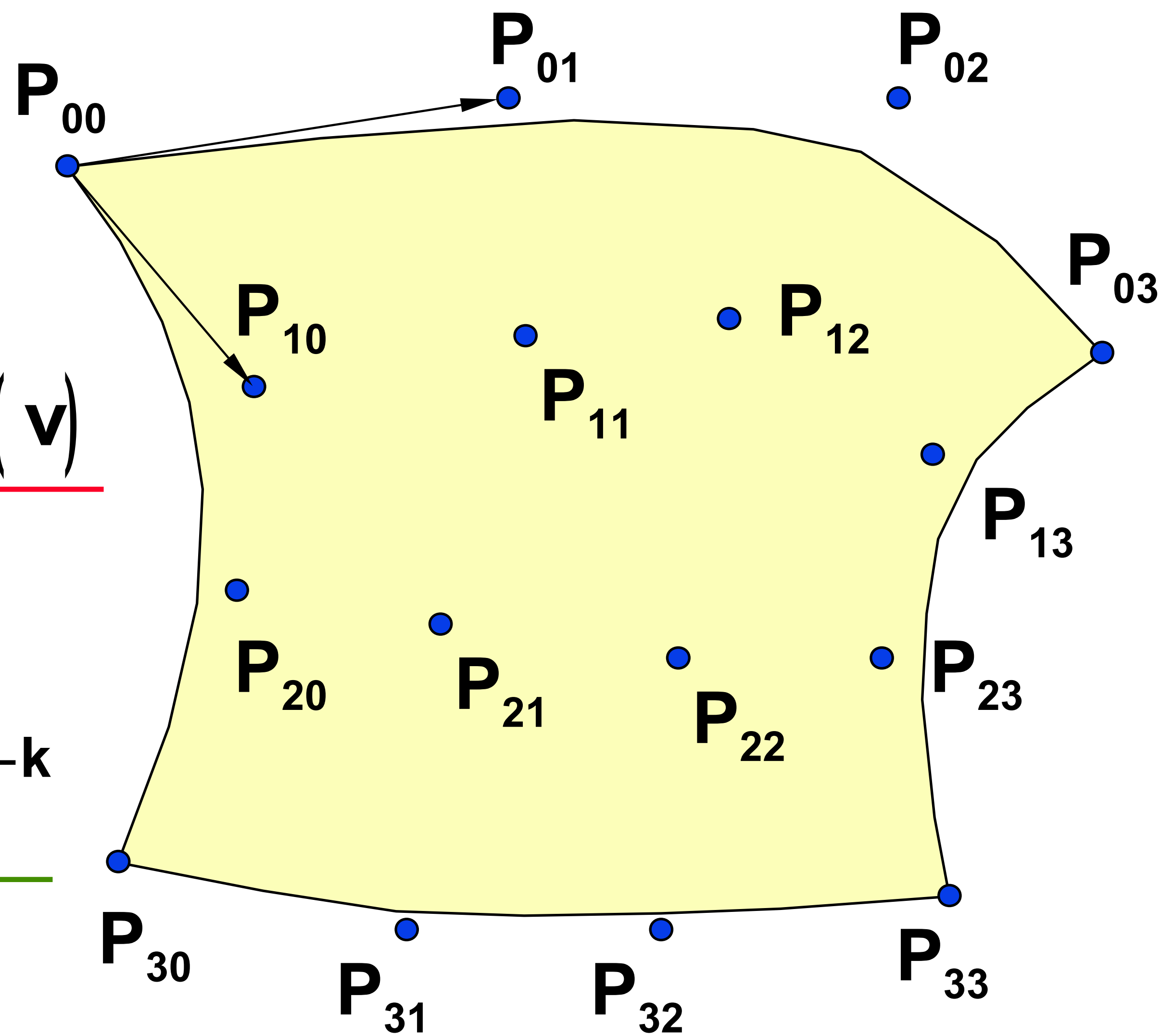
$$\mathbf{P} = \left[P_{ij} \right]_{i,j=0}^{3}$$

$$P(u,v) = \mathbf{B}(u)^T \cdot \mathbf{P} \cdot \mathbf{B}(v)$$

$$\mathbf{B}(t) = \left[B_k(t) \right]_{k=0}^{3}$$

$$B_k(t) = \binom{3}{k} t^k (1-t)^{3-k}$$

Bernstein polynomials



Bernstein polynomials

- $\mathbf{B_k(t)}$ are **nonnegative cubic** polynomials for

  $\mathbf{k = 0\ ..\ 3}$ and $\mathbf{0 \le t \le 1}$

- $\mathbf{\sum_k B_k(t) = 1}$ for arbitrary $\mathbf{t}$
  - Cauchy's condition (affine invariance)

➡ if $\mathbf{B_k(t)}$ are used as weight coefficients (linear blending), result will be in a **convex hull** of input data (control polygon vertices in this case)

- $\mathbf{B_k(t)}$ are blending coefficients of a convex combination

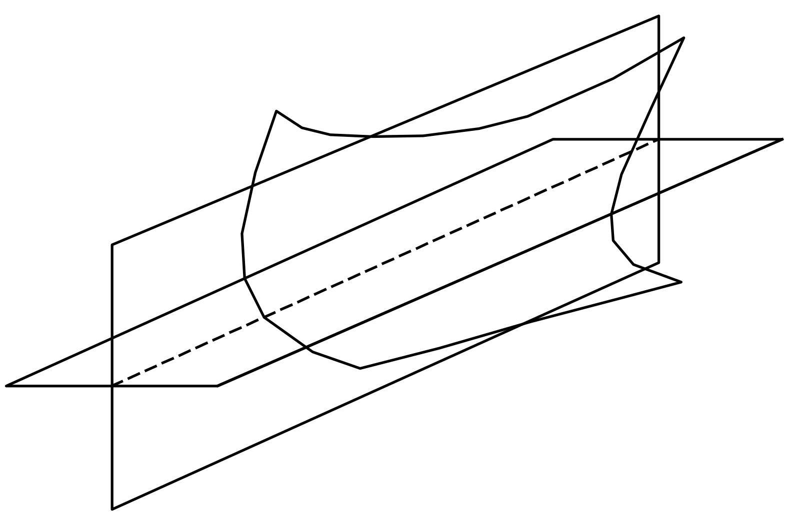
Ray vs. Bèzier patch intersection

- after converting a bicubic Bèzier patch to implicit form we've got an **algebraic surface of the 18th degree !**
  - **18th degree polynomial** to solve

- $\mathbf{B(u,v) = P_0 + t \cdot \vec{p}_1}$ is an algebraic system, three equations for three quantities: **t, u, v**
  - can be solved using 3D **Newton iteration** (converges only in a relatively small interval)

Ray vs. Bèzier patch II

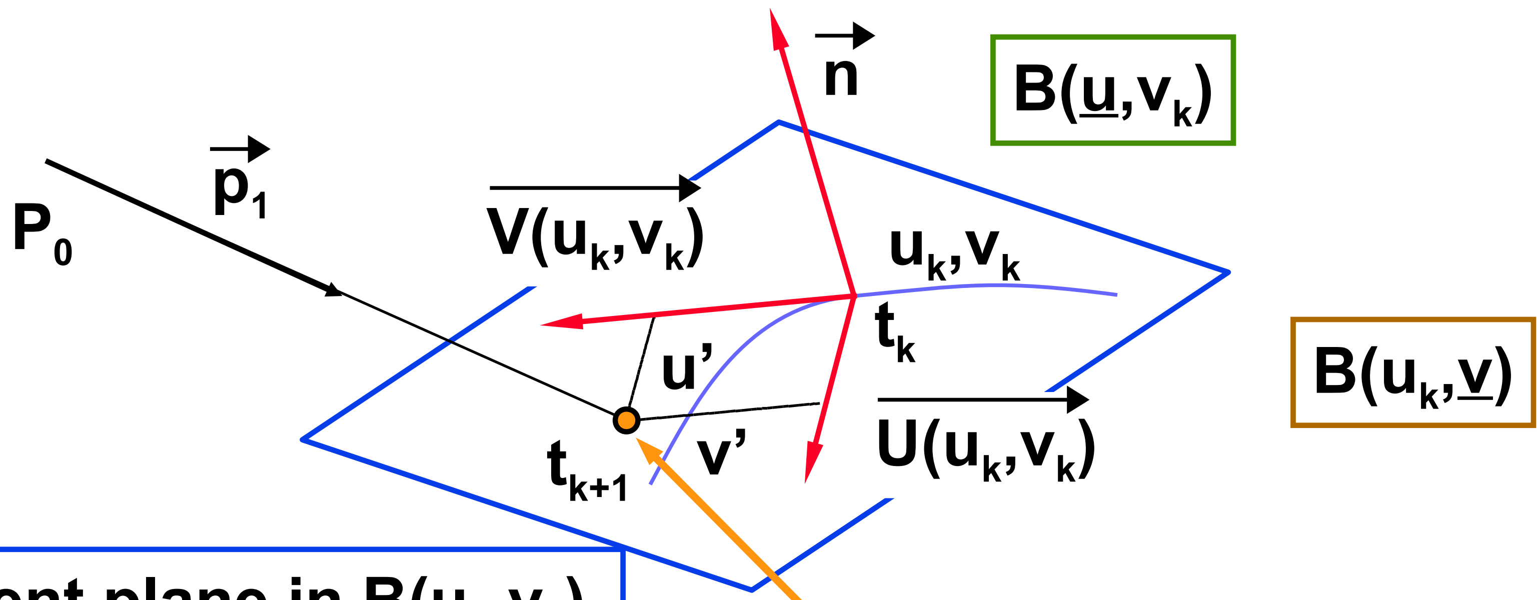
- system of 2 algebraic equations for 2 quantities **u**, **v**:
  - **t** can be eliminated from the previous system
  - let ray be **intersection of two planes**, planes vs. Bèzier pach are examined
  - solution by a 2D **Newton iteration**



$$F_1(u,v) = 0$$

$$F_2(u,v) = 0$$

3D “Newtonian” iteration



tangent plane in $B(u_k,v_k)$

Ray × tangent plane intersection: t_{k+1}, u’, v’

$$V\left(u_k, v_k\right) = \frac{\partial B}{\partial v}\left(u_k, v_k\right)$$

$$U\left(u_k, v_k\right) = \frac{\partial B}{\partial u}\left(u_k, v_k\right)$$

$$u_{k+1} = u_k + u'$$
$$v_{k+1} = v_k + v'$$

Bèzier patch subdivision

- one Bèzier patch **B(u,v)** $[\, 0 \le u,v \le 1 \,]$ can be divided into four smaller ones:

  $\mathbf{B_{00}(u,v)}$ $[\, 0 \le u,v \le 1/2 \,]$

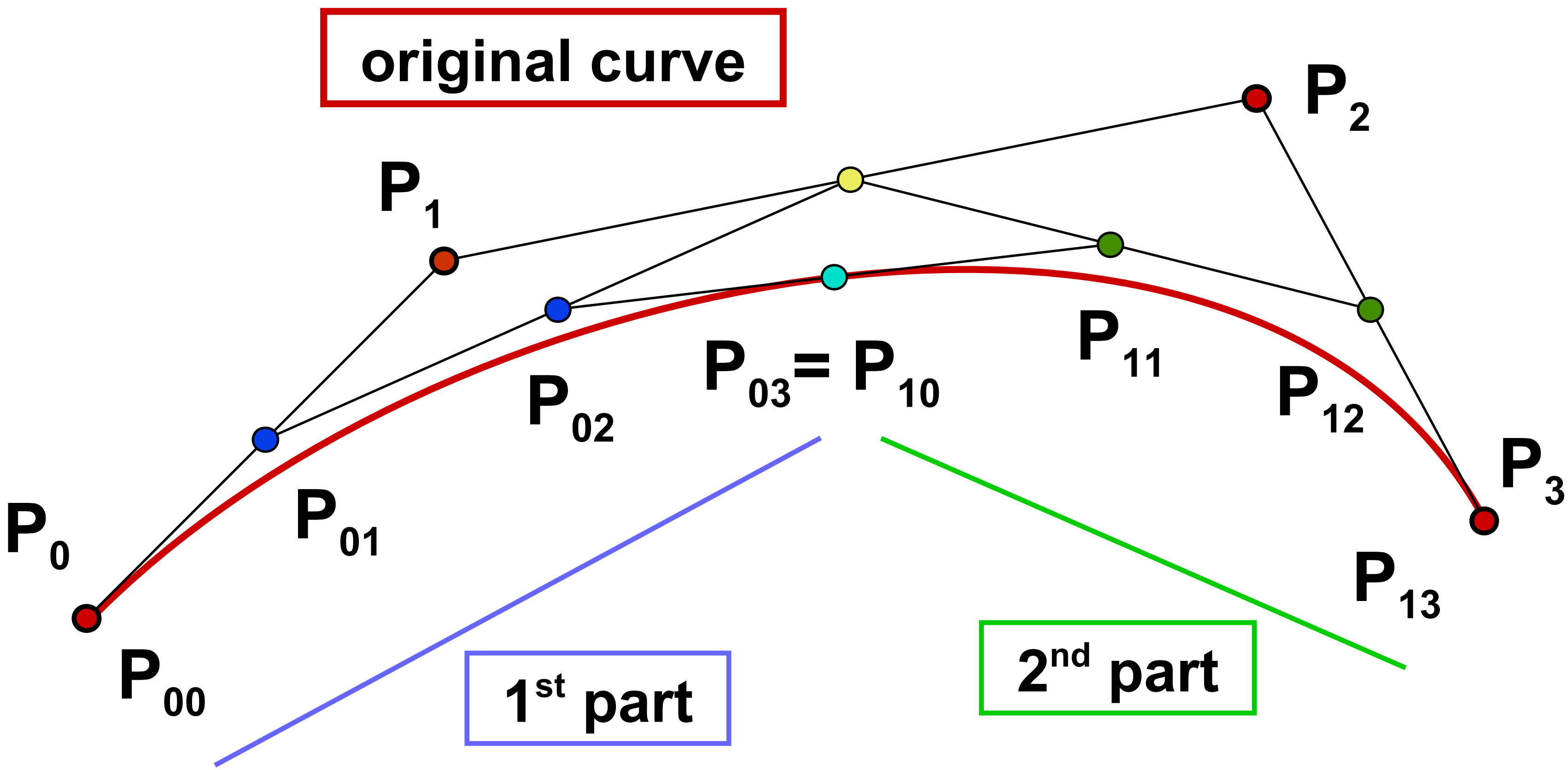
  $\mathbf{B_{01}(u,v)}$ $[\, 0 \le u \le 1/2,\ 1/2 \le v \le 1 \,]$

  $\mathbf{B_{10}(u,v)}$ $[\, 1/2 \le u \le 1,\ 0 \le v \le 1/2 \,]$

  $\mathbf{B_{11}(u,v)}$ $[\, 1/2 \le u,v \le 1 \,]$

- ➡ new control points can be computed using recursive algorithm of **P. de Casteljau**
  - only addition and dividing by two is used in this case!

De Casteljau subdivision (2D)



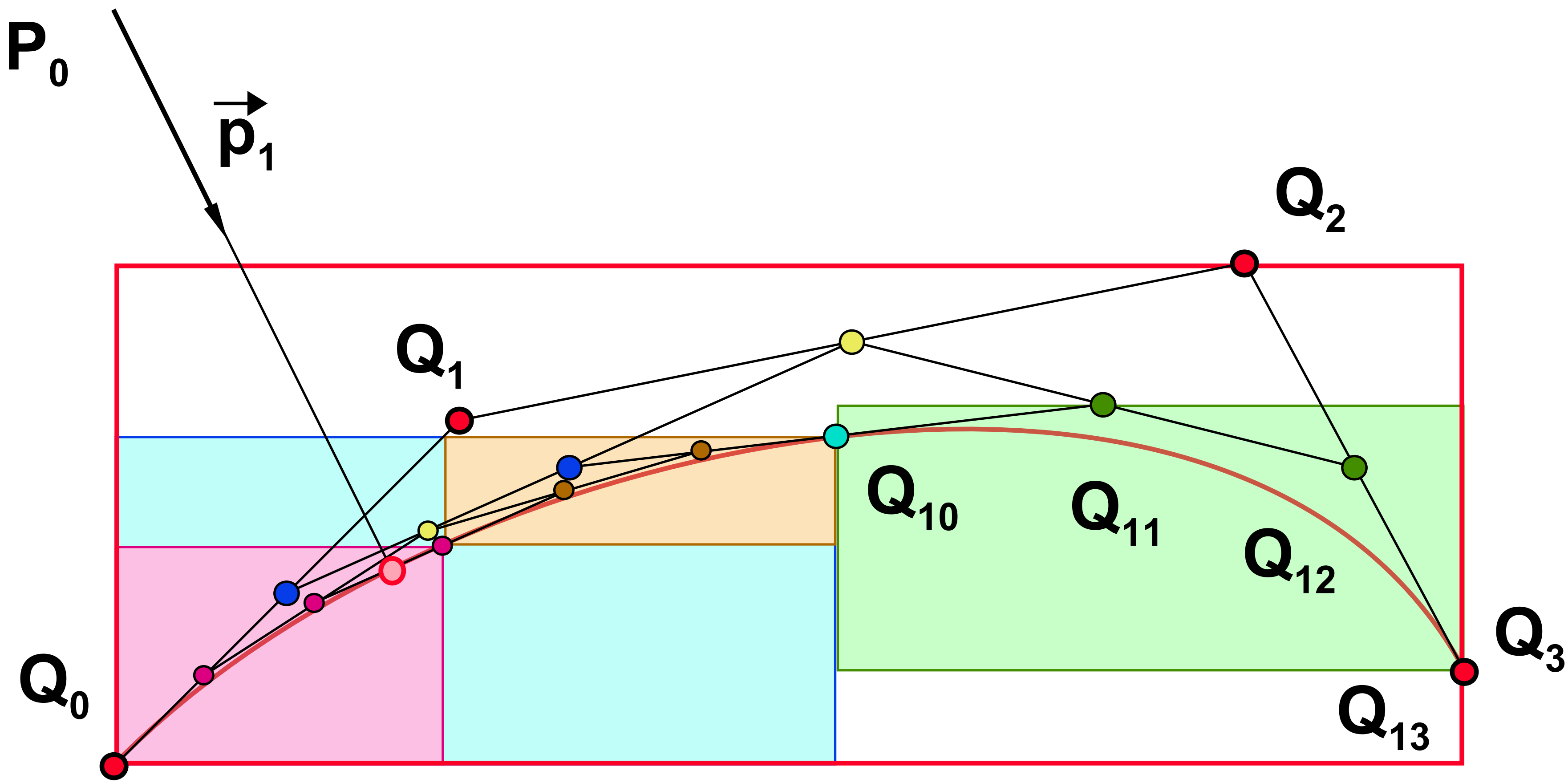
Algorithm ideas

- we are looking for the **closest intersection** of the ray with the **set of Bèzier patches**

- every **Bèzier patch** lies inside a **convex hull** of its control points
  - we will store **bounding box** of every patch (x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})

- relevant patch will be subdivided as long as it is intersected by a ray and too large to start the **Newtonian iteration** in it
  - criterion: small **surface curvature**

Bounding boxes



Algorithm outline

❶ intersected **bounding boxes** are maintained in the order of the intersection (**front-to-back**) .. heap

❷ the closest bounding box is selected: if it has proper (low) curvature, the **Newtonian iteration** is started in it. If an actual intersection is found, it is placed into the result set.

– the whole algorithm ends if the closest intersection is closer that the closest unprocessed patch (box)

❸ the closest patch with **high curvature** is divided into four parts, they are reinserted into the list (heap)

– go back to ❷

References

- **A. Glassner**: ***An Introduction to Ray Tracing***, Academic Press, London 1989, 99-102
- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 507-528