Parametrization of Catmull-Clark Subdivision Surfaces for Posture Generation

Adrien Escande, Stanislas Brossette, Abderrahmane Kheddar

▶ **To cite this version:**

Adrien Escande, Stanislas Brossette, Abderrahmane Kheddar. Parametrization of Catmull-Clark Subdivision Surfaces for Posture Generation. ICRA: International Conference on Robotics and Automation, May 2016, Stockholm, Sweden. pp.1608-1614, 10.1109/ICRA.2016.7487300 . hal-01275693

HAL Id: hal-01275693
https://hal.science/hal-01275693

Submitted on 18 Feb 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Parametrization of Catmull-Clark Subdivision Surfaces for Posture Generation

Adrien Escande[1], Stanislas Brossette[1,2] and Abderrahmane Kheddar[1,2]

***Abstract*— In this paper we propose a method to build a smooth and non-singular map from the unit sphere to a Catmull-Clark subdivision surface. We use a tailored ray-casting algorithm to associate a point of the surface to each point of the sphere. This allows us to have smooth approximate representations for a large variety of (possibly non-convex) meshes that we can use to write numerically well-behaved constraints in gradient-based optimization. In particular, we address the writing of contact constraints between a robot and objects of its environment. We first detail our algorithm, taking care of speed and robustness, then show several use-cases for posture generations with a humanoid robot or a hand.**

I. Introduction

In the control and planning of complex robots, it is sometimes needed to find a configuration of the robot satisfying a set of constraints [1] [2] [3]. For example, for a humanoid robot, one might want to find a configuration where the two feet are on a surface and the hand reaches an object, while ensuring the robot is stable, within its joint and torque limits, and free of collisions. We refer to this configuration posture problem as *Posture Generation* (PG), and it can be formulated as a non-linear constrained optimization problem and solved by adequate numerical algorithms [4].

For underactuated robots such as humanoid robots, contact is an important notion, as it is the only mean to move and maintain stability. Symmetrically, contact plays an important role in manipulation, where the manipulated object is underactuated and is moved and maintained stable by the robot. Writing constraints expressing contacts is thus a central part of posture generation. For contacts between a body of the robot and simple geometry forms (planes, spheres, cylinders, ...), ad hoc constraints can be written easily. They are sufficient to tackle a large number of interesting scenarios, especially for robots in structured environments [3]. However, contacts with more complex objects is needed too, for unstructured environment or for manipulating objects.

One extension is to express contact with parametric surfaces (see [5] for an example): given a surface defined by the function $(u, v) \in [0, 1]^2 \mapsto s(u, v) \in \mathbb{R}^3$ and a point p fixed on a body of a robot, we can write the constraints[1]

$$s(u, v) = p(q) \tag{1}$$

$$\nu_s(u, v) = -\nu_p(q) \tag{2}$$

*This work was supported by the H2020 Europen project COMANOID

[1]CNRS-AIST Joint Robotics Laboratory (JRL) UMI3218/RL

[2]CNRS-UM LIRMM

[1]note that in practice, for a better numerical behavior of the optimization, we write that ν_s is perpendicular to the tangent plane of the robot surface at p and has opposite direction with ν_p, instead of simply $\nu_s(u, v) = -\nu_p(q)$.

where q is the configuration of the robot, and ν_s (resp. ν_p) is the outer normal vector to the parametric surface (resp. the robot's surface) at $s(u, v)$ (resp. p). All quantities are expressed in the same frame. The variables u and v become parameters of the optimization problem. To work with derivative-based optimization algorithms, such as Sequential Quadratic Programming (SQP) or Interior Points (IP), which are much faster than derivative-free algorithms, the functions involved in an optimization problem need to be at least C^1 and gradients must be full rank at the solution (for ensuring constraints qualification, see [6]). This requests s and nu_s to be at least C^1, with respect to (w.r.t.) (u, v) and since the solution can be anywhere, we need their derivatives to be always full rank.

This approach has two limits: (i) for a given object, a parametrization of its surface is rarely available, especially a parametrization conforming with the continuity requirement. On the contrary, we usually have an approximation of the object as a mesh; (ii) for surfaces of closed objects topologically like a sphere, which are a large part of the objects of interest in applications, a parametrization from $[0, 1]^2$ with full rank and continuous gradient does not exist, because there is no diffeomorphism between the unit sphere and a subset of $\mathbb{R}^2$.

The contribution of this paper is to provide a systematic way to obtain a parametric surface closely approximating a mesh and meeting the regularity requirements mentioned above, for a large class of objects, namely star-convex objects. To that end, we combine the use of Catmull-Clark subdivision surfaces with a tailored ray casting algorithm, and propose an efficient and robust numerical method to evaluate a point, a normal and their derivatives at a given set of parameters.

The paper is organized as follows: we first give a very brief overview of Catmull-Clark subdivision surfaces and of the method we use to evaluate them (Sec. II), then propose our parametrization (Sec. III). We then describe the computations of the normal vector (Sec. IV) and the derivatives (Sec. V), carefully addressing possible numerical issues. In Sec. VI, we provide a mean to speed up the computations, before giving timings and showing several examples where we use constraints (1)-(2) with the proposed parametrization (Sec. VII). We then offer some perspectives (Sec. VIII).

II. Catmull-Clark subdivision surfaces

With the years, Catmull-Clark subdivision surfaces (CCS), first introduced in 1978, have become a powerful tool for

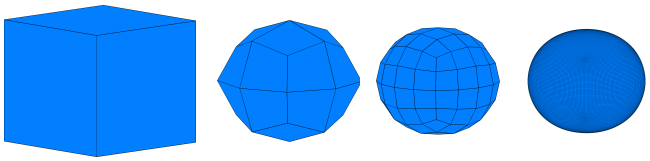

Fig. 1. Several Catmull-Clark subdivisions of a cube. From left to right: original mesh, after 1, 2 and 6 iterations

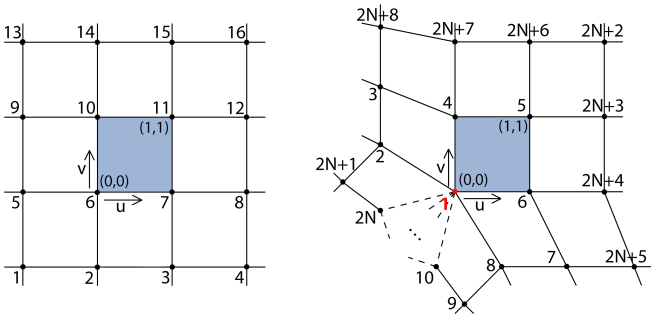

Fig. 2. Left: a regular face (in blue) with its local u-v coordinates and the ordering of the surrounding vertices. Right: a face (in grey) containing a extraordinary vertex with valence N (in red).

modeling, and have been widely adopted in computer graphics. For a given initial mesh, to which we will refer as *control mesh*, the corresponding CCS is the limit surface obtained when applying iteratively the following subdivision scheme:

1) for each face, create a *face point* defined as the average of the vertices of the face,
2) for each edge, compute its middle point m_e, the middle point m_f of the two newly computed *face points* corresponding to the adjacent faces, and create an *edge point* $p_e = \alpha_e m_e + \beta_e m_f$,
3) for each vertex v, compute the average a_f of the face points corresponding to the faces the vertex belongs to, the average a_e of the edge points corresponding to the edges the vertex belongs to, and create a *vertex point* $p_v = \alpha_v a_f + \beta_v a_e + \gamma_v v$
4) for each (original) edge, create an edge between the edge point and each neighbour vertex point and face point.
5) The created points in 1-3, the edges created in 4 and the faces they define give the subdivided mesh.

The parameters for CCS are $\alpha_e = \beta_e = 1/2$, $\alpha_v = 1/N$, $\beta_v = 2/N$ and $\gamma_v = (n-3)/N$ where N is the *valence* of vertex v, *i.e.* the number of edges the vertex belongs to. Fig. 1 illustrates the process. Note that new vertices are obtained by linear combination of the old ones.

It was shown by Stam in a seminal paper [7] that points on CCS can be computed by direct analytical formula without explicitly subdividing the control mesh. In the remainder of this section, we recall Stam's methodology.

A vertex whose valence is 4 is called an *ordinary* vertex. Otherwise it is coined *extraordinary*. We can suppose that each face of the control mesh is a quadrilateral and contains at most one extraordinary vertex. If it is not the case, subdividing at most twice the mesh with the above scheme and taking the result as the new control mesh will ensure

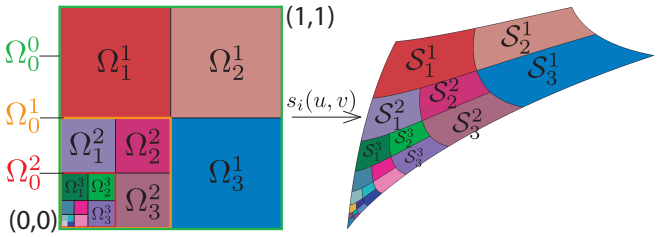

Fig. 3. Mapping from the unit square Ω_0^0 and its subdivision to a surface patch. $\mathcal{S}_k^n$ is the image of Ω_k^n by s_i.

these properties while yielding the same limit surface. To each face of the control mesh we can associate a surface patch which is the limit of the subdivisions for this face. The CCS is the union of all patches. Each patch can be represented as a parametric surface and we now explicit the function $s_i : (u,v) \in [0,1]^2 \mapsto s_i(u,v) \in \mathbb{R}^3$ corresponding to the i-th face.

For a regular face, *i.e.* a face with no extraordinary vertex, the patch is simply a bi-cubic B-spline, depending solely on the 16 vertices surrounding the face (see Fig. 2). Therefore, there is no need to further subdivide this face to compute its patch. Given the 16-by-3 matrix C_i regrouping the coordinates of those 16 points ordered as Fig. 2 (left), we simply have $s_i(u,v) = C_i^T b(u,v)$ where $b(u,v)$ is the vector of the b-spline basis obtained by concatenating the columns of $B(u,v)$:

$$B(u,v) = \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}^T M^T M \begin{bmatrix} 1 \\ v \\ v^2 \\ v^3 \end{bmatrix}, \text{ with } M = \frac{1}{6}\begin{bmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For a non-regular face, the idea is that one subdivision replaces the face by 3 regular faces, which can be evaluated as above and one non regular one which need to be subdivided further. More precisely, let $\Omega_0^0 = [0,1]^2$ and define an infinite partition of it as illustrated in Fig. 3, for $n = 1..\infty$:

$$\Omega_0^n = \left[0, 2^{-n}\right]^2, \Omega_1^n = \left[0, 2^{-n}\right] \times \left[2^{-n}, 2^{-n+1}\right],$$
$$\Omega_2^n = \left[2^{-n}, 2^{-n+1}\right]^2, \Omega_3^n = \left[2^{-n}, 2^{-n+1}\right] \times \left[0, 2^{-n}\right]$$

We denote by $\mathcal{S}_k^n$ the surface image of Ω_k^n by s_i. For each n we have $\Omega_0^{n-1} = \Omega_0^n \cup \Omega_1^n \cup \Omega_2^n \cup \Omega_3^n$, and likewise for $\mathcal{S}_0^{n-1}$.

We choose the parametrization so that $(0,0)$ corresponds to the extraordinary vertex. To evaluate $\mathcal{S}_0^0$, we need the $K = 2N + 8$ vertices surrounding the face (see Fig. 2, right). Let us denote by $C_{i,0}$ the K-by-3 matrix containing in each row the coordinates of these vertices. To evaluate $\mathcal{S}_0^1$, we need K vertices as well, whose matrix of coordinates C_1 can be computed as $A_N C_{i,0}$, where A_N is the K-by-K matrix expressing locally the subdivision process. To evaluate $\mathcal{S}_k^1$, $k = 1..3$, we need 9 more vertices, obtained from C_0 as $\bar{C}_{i,1} = \bar{A}_N C_{i,0}$. By recursion, the coordinates of the vertices needed to evaluate $\mathcal{S}_0^n$ (resp. $\mathcal{S}_k^n$, $k = 1..3$) are given by $C_{i,n} = A_N^n C_{i,0}$ and (resp. $\bar{C}_{i,n} = \bar{A}_N A_N^{n-1} C_{i,0}$).

For $n \geq 1$, the surfaces $\mathcal{S}_k^n$, $k = 1..3$ correspond to regular faces of the mesh obtained after n subdivisions of

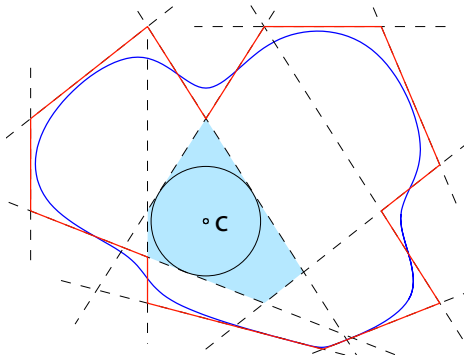

Fig. 4. 2d illustration of a control mesh (in red) and the CCS (in blue). The dashed lines depict the supporting planes and define the kernel (light blue). c is the center of the largest sphere (black circle) in it.

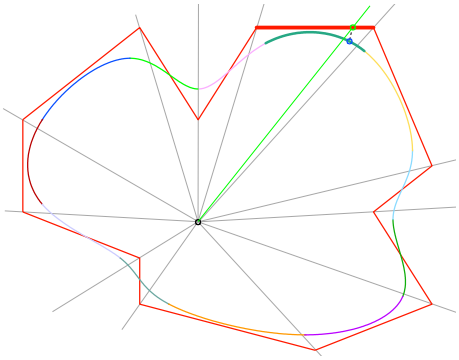

Fig. 5. Patches corresponding to the different faces of the control mesh are depicted in different colors. The cones are delimited by the gray lines. The casted ray (in green)intersects a face (in bold red). From the intersection point (in green) a point (in blue) is deduced on the corresponding patch.

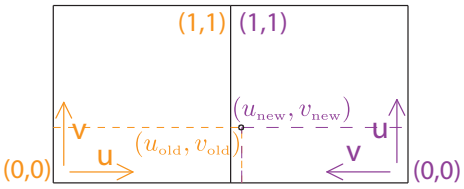

Fig. 6. Example of coordinates change when changing from the left patch to the right one. With the depicted frames, the change is $u_{\text{new}} = v_{\text{old}}$ and $v_{\text{new}} = 2 - u_{\text{old}}$. There are in total 16 different changes of coordinates depending on the relative orientation of the frames.

the control mesh, and therefore can be evaluate as above: the 16 coordinates are extracted from $\bar{C}_n$ by a selection matrix P_k, and we have the restriction of s_i to Ω_k^n:

$$s_i(u,v)\big|_{\Omega_k^n} = \bar{C}_n^T P_k^T b(t_{n,k}(u,v)) \tag{3}$$

where $t_{n,k}$ maps Ω_k^n to $[0,1]^2$: $t_{n,1}(u,v) = (2^n u - 1, 2^n v)$, $t_{n,2}(u,v) = (2^n u - 1, 2^n v - 1)$ and $t_{n,3}(u,v) = (2^n u, 2^n v - 1)$.

The last step of Stam's work is to make use of the eigen decomposition $A_N = V\Lambda V^{-1}$ where $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_K)$: $\bar{C}_{i,n}$ can be computed efficiently as $\bar{A}_N V \Lambda^{n-1} V^{-1} C_0$. Noting $\hat{C}_{i,0} = V^{-1} C_0$, and $x_N(u,v,k) = (P_k \bar{A}_N V)^T b(u,v)$, eq. (3) can be rewritten, with a bit a algebra

$$s_i(u,v)\big|_{\Omega_k^n} = \sum_{j=1}^{K} \lambda_j^{n-1} x_{N,j}(t_{n,k}(u,v),k) p_j \tag{4}$$

with p_j the j-th column of $\hat{C}_{i,0}$. Since $\lambda_1 = 1$ and all $0 < \lambda_i < 1$ for $i = 2..K$, $s_i(0,0)$ is simply p_1 ($x_{N,1}(0,0,k) = 1$ by taking the limit).

Because A_N depends solely on the valence N, its eigen decomposition can be done once and for all, and $\hat{C}_{i,0}$ needs only be computed once for a given control mesh.

III. Ray-casting base parametrization

For a mesh with n_f faces, the previous section gives n_f function s_i which are local parametrization of the CCS. In this section we propose a global C^1 parametrization with full-rank gradient. We are interested in closed object and we know it is not possible to get such a parametrization over a subset of $\mathbb{R}^2$, therefore we propose a parametrization over S^2 the unit sphere in $\mathbb{R}^3$. The general idea is to take a point c inside the object and to associate to each unit vector d the point of the object on the ray $c + td,\ t > 0$. This is akin to computing the inverse of the CCS projection onto a sphere centered at c.

A. Determination of a center point

For every ray to correspond to a single point of the surface, the segment from c to any point of the surface must remain inside the object. By definition, such a point c exists if and only if the CCS defines a star-convex volume.

Each face of the control mesh is made of two triangles (that need not be coplanar), each of which defines a half-space on the side of its supporting planes indicated by the inward normal. The intersection of all these half-spaces is named the *kernel* of the mesh (see *e.g.* [8, Ex. 33.3-4]) and is the locus of all points seeing all the surface of the control mesh (Fig. 4).

We use the following heuristic to get c: we compute the Chebyshev center of the above kernel (i.e. the center of the largest ball inside the kernel). This is done simply by linear programming: given the half-spaces described as $a_i^T x \le b_i$, we solve (see [9, Sec. 4.3.1])

$$\begin{aligned} \max_{c,r} \quad & r \\ \text{s.t.} \quad & a_i^T c + \|a_i\| r \le b_i,\ \ i = 1..2n_f \end{aligned} \tag{5}$$

This particular choice of c is motivated by robustness considerations as detailed in next subsection.

Since the computation of c is based on the control mesh and not on the CCS, we could devise extreme examples where c is not suitable. However, this works well in practice especially because to get isolated extraordinary vertices we subdivide twice the original mesh so that the actual control mesh is close to the final CCS. Should it be needed, we could subdivide the control mesh a few times before computing c.

B. Central ray-casting algorithm

To determine the intersection of a ray $c+td$ with the CCS, we borrow the idea from ray-casting in computer graphics. For a parametric surface, one idea (see [10]) is to solve numerically in (u,v,t) the equation $c + td = s(u,v)$ with $t > 0$. In the general case rays can come from anywhere and there is a great deal of refinements to ensure the robustness

of the scheme. This is not our case however, since we have the particularity of casting rays from inside the object and are guaranteed to have a single solution to the equation.

Assume we know which patch i the ray intersects. Then we know that $t > 0$ and we simply solve the equation $f_i(x) = 0$ by Newton method, where $x = \begin{bmatrix} u & v & t \end{bmatrix}$ and $f_i(x) = s_i(u,v) - c - td$. In practice we do not know i beforehand and we need to provide an initial guess $x_0 = \begin{bmatrix} u_0, v_0, t_0 \end{bmatrix}$ close to the solution for the method to converge.

Since the CCS is not far from the control mesh, we take i to be the index of the face of the control mesh intersected by our ray. The test to check if the ray intersects a face is very simple: the point c and the 4 points of a face defines a cone of apex c, see Fig. 5. Each side of the cone has an outward normal vector $n^c_{i,j}$ and we can define $A_i^T = \begin{bmatrix} n^c_{i,1} \cdots n^c_{i,4} \end{bmatrix}$. There is intersection if $A_i d \leq 0$. A basic way to find i is thus to test the previous condition for each face in order. We give a more efficient solution in Section VI.

Once we know the face, we compute its intersection point with the ray, and retrieve the (u,v) coordinates in the face of this point by a perspective mapping of the face onto a unit square (see [11]). This gives us our initial guess x_0.

The patch intersected by the ray may not correspond to the face we found. This is easily detected and taken care of during the iterations of the Newton method: if at an iteration the uv part of the current x is not in $[0,1]^2$, we change to the adequate neighbour patch, estimate the uv in this new patch, and resume the computation. By adequate, we mean that if for example $u > 1$ we change to the patch on the other side of the edge corresponding to $u = 1$. To estimate the new uv, we proceed as if both neighbour faces were coplanar unit square and make a simple change of coordinates, see Fig. 6. If the new uv coordinates are still out of $[0,1]^2$ (because both were originally out), we repeat the process.

The overall algorithm is summarized by Alg. 1. From a vector d, it returns the index i of the patch intersected by the ray $c + td$, and the triplet $x(d) = (u(d), v(d), t(d))$ localizing the intersection point $p(d) = s_i(u,v) = c + td$ on the patch and on the ray. For practical implementation, a safeguard must be added to avoid infinite cycles (which we never encountered). We could also add a line search to prevent the Newton iteration to diverge. In practice however, the initial guess is sufficiently close to the solution so that unit steps can be taken and the convergence is quadratic: we typically need $3-4$ iterations to get a norm of residual below $\epsilon = 10^{-14}$. The Jacobian matrix of $\partial f_i / \partial x$ is

$$J_i = \frac{\partial f_i}{\partial x_i} = \begin{bmatrix} \delta_{u,i} & \delta_{v,i} & -d \end{bmatrix}, \text{ with } \delta_{y,i} = \frac{\partial s_i}{\partial y}, \ y = u, v$$

It can be badly conditioned in three situations: (i) the derivative in u and v diverge, which happens close to extraordinary vertices with Stam's proposed parametrization, (ii) d is (almost) in the subspace spanned by $\delta_{u,i}$ and $\delta_{v,i}$ which is the tangent plane to the patch at the current (u,v), or (iii) the patch is orders of magnitude longer than large so that $\frac{\partial s_i}{\partial u}$ and $\frac{\partial s_i}{\partial v}$ are almost colinear. Situation (i) can occur only if the initial guess is extremely close to the extraordinary vertex (say $u, v \leq 10^{-30}$), in which case we perturb slightly the guess (*e.g.* take $u = v = 10^{-16}$). The iterations will never come back to such an extreme case because convergence will be reached before. Situation (ii) is avoided by our choice of c. Situation (iii) should rarely occur and can be handled by pre-processing of the mesh.

Algorithm 1 $(i, x) = \text{raycasting}(d)$

```
i ← findFace(d)
x ← initialGuess(i, d)
e ← f_i(x)
while ‖e‖ > ε do
    x ← x − (∂f_i/∂x)^{-1} e        //Newton step
    if x ∉ [0,1]^2 × ℝ then
        i_new ← neighbour(i, x)
        x ← changeCoordinates(i, i_new, x)
        i ← i_new
    end if
    e_i ← f_i(x)
end while
return (i, x)
```

IV. Normal vector computation

Given a patch i and local coordinates (u,v), the normal vector $\nu_i(u,v)$ is obtained by computing the cross product $\delta_{u,i} \times \delta_{v,i}$ and normalizing it, as is classical with parametric surfaces. Thus for our global parametrization, the normal vector $\nu(d)$ is obtained by first determining $i(d)$, $u(d)$ and $v(d)$ with Alg. 1 then computing $\nu(d) = \nu_{i(d)}(u(d), v(d))$.

For regular patches, the computation is straightforward. For non-regular patches, care must be taken in the computations to avoid numerical issue when nearing $(0,0)$. Indeed, we have

$$\delta_{u,i}(u,v)\big|_{\Omega^n_k} = 2^n \sum_{j=1}^{K} \lambda_j^{n-1} \frac{\partial}{\partial u} x_{N,j}(t_{n,k}(u,v), k) p_j \quad (6)$$

(and likewise for v), where the 2^n comes from the derivative of $t_{n,k}$. It was observed in [12] that the first term of this sum is 0, and it is important to not compute it (*i.e.* start the sum at $j = 2$) because numerically it will not be exactly 0 and the error will be multiplied by a potentially huge 2^n.

We can suppose without loss of generality that the eigenvalues are given in decreasing orders. Whatever N, we have $\lambda_2 = \lambda_3 > 0.5$ (see [7]), so that the $\delta_{u,i}$ and $\delta_{v,i}$ diverge when (u,v) goes to $(0,0)$. However, since we are normalizing the cross product to get the normal, the magnitude of the $\delta_{u,i}$ and $\delta_{v,i}$ is not important. In particular, we can divide them by $\kappa_n = 2^n \lambda_2^{n-1}$ so that with $\delta'_{y,i} = 1/\kappa_n \delta_{y,i}$,

$$\nu_i(u,v) = \frac{\delta'_{u,i} \times \delta'_{v,i}}{\left\| \delta'_{u,i} \times \delta'_{v,i} \right\|} \quad (7)$$

and all terms stay bounded an non-zero (when (u,v) tends to $(0,0)$, $\delta'_{u,i}$ behaves like $\frac{\partial}{\partial u} x_{N,2}(t_{n,k}(u,v), k) p_2 + \frac{\partial}{\partial u} x_{N,3}(t_{n,k}(u,v), k) p_3$, likewise for $\delta'_{v,i}$, and the x_N

derivatives are bounded and non-zero). Because CCS are tangent plane continuous (see [12]), $\nu_i(u,v)$ converges in $(0,0)$.

V. Derivatives

In this section, we compute the derivatives $\partial(p)/\partial(d)$ and $\partial(\nu)/\partial(d)$, and study their properties. Since i is known for a given d we drop it in the notation for conciseness.

First we compute the derivatives of $(u(d), v(d), t(d))$ w.r.t. d. $(u(d), v(d), t(d))$ verify the relation $s(u,v) - c - td = 0$. Deriving this relation w.r.t. d, we get

$$\begin{bmatrix}\delta_u & \delta_v - d\end{bmatrix}\begin{bmatrix}\partial u/\partial d \\ \partial v/\partial d \\ \partial t/\partial d\end{bmatrix} = tI \tag{8}$$

from which we can compute the derivatives w.r.t. d and define

$$\begin{bmatrix}\partial u'/\partial d \\ \partial v'/\partial d \\ \partial t/\partial d\end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix}\kappa_n \partial u/\partial d \\ \kappa_n \partial v/\partial d \\ \partial t/\partial d\end{bmatrix} = t\begin{bmatrix}\delta'_u & \delta'_v - d\end{bmatrix}^{-1} \tag{9}$$

$\begin{bmatrix}\delta'_u & \delta'_v - d\end{bmatrix}$ is well conditioned and bounded so that $\partial u'/\partial d$, $\partial v'/\partial d$ and $\partial t/\partial d$ are bounded and non-vanishing.

We get immediately

$$\frac{\partial p}{\partial d} = \begin{bmatrix}\delta_u & \delta_v\end{bmatrix}\begin{bmatrix}\partial u/\partial d \\ \partial v/\partial d\end{bmatrix} = \begin{bmatrix}\delta'_u & \delta'_v\end{bmatrix}\begin{bmatrix}\partial u'/\partial d \\ \partial v'/\partial d\end{bmatrix} \tag{10}$$

(the second form yields more stable computations) which is continuous because all the quantities are (CCS are C^2 except at extraordinary vertices where they are C^1). As a product of rank 2 matrices, it is however not full rank. This is because $p(d) = p(\alpha d)$ for $a > 0$. But for the same reason, if p is seen as an application from S^2 to $\mathbb{R}^3$, its derivative w.r.t a local parametrization of S^2 is full rank.

The derivative of the normal is given (for $i = 1..3$) by

$$\frac{\partial \nu}{\partial d_i} = \frac{1}{\|\delta'_u \times \delta' v\|}\left(I - \nu\nu^T\right)\left(\frac{\partial \delta'_u}{\partial d_i} \times \delta'_v + \delta'_u \times \frac{\partial \delta'_v}{\partial d_i}\right) \tag{11}$$

It is continuous everywhere but at extraordinary vertices where it diverges. The same discussion about rank as above applies.

VI. Accelerating the face identification

In section III, we proposed a simple test to find which face of the control mesh is intersected by the ray $c + td$, $t > 0$. This test takes most of the time in Alg. 1 (typically 75% of the time for n_f around 5000). We present a cheaper alternative.

We denote $\mathcal{C}_i$ the cone defined by the center c and the i-th face of the control mesh, as described in Sec. III. We have $\mathcal{C}_i = \left\{x \in \mathbb{R}^3, A_i x \leq 0\right\}$. For each vertex $v_{i,j}$, $j = 1..4$ belonging to the face, we define $r_{i,j} = v_{i,j} - c$ the vectors supporting the edges of the cone and denote $\hat{r}_i$ their average. Finally, we call *opposite* to $\mathcal{C}_i$ and note $\bar{\mathcal{C}}_i$ the unique cone $\mathcal{C}_j$ containing $-\hat{r}_i$ (note that this is not a one-to-one relationship).

Given a direction d, we identify the intersected face with the algorithm given in Alg. 2: we start from an initial face i (by default $i = 0$ or random), and check if d is in $\mathcal{C}_i$. If it is not, we check if $\mathcal{C}_i$ intersects the open half-space $\mathcal{H}_d$ defined by d. This is the case if and only if $d^T r_{i,j} > 0$ for at least one j. If not, we go to the opposite of $\mathcal{C}_i$ which then necessarily intersects $\mathcal{H}_d$: all vectors $-r_{i,j}$ are in $\mathcal{H}_d$, thus $-\hat{r}_i$ is in it as well, so that by definition, $-\hat{r}_i \in \bar{\mathcal{C}}_i \cap \mathcal{H}_d \neq \emptyset$. From there we iteratively pass from one cone to the other by choosing each time the adjacent cone indicated by the line of $A_i d$ with the largest value, *i.e.* the cone on the other side of the (cone's) face for which the condition $A_i d \leq 0$ is the most violated. We loop until i is such that $A_i d \leq 0$.

Algorithm 2 $i = \text{findFace}(d, i_0)$

```
i ← i_0
if A_i d ≤ 0 then
    return i
else
    if d^T r_{i,j} ≤ 0 for all j = 1..4 then
        i ← opposite(i)
    end if
    while A_i d > 0 do
        j ← arg max._j (A_i d)_j
        i ← neighbour(i, j)
    end while
end if
return i
```

The matrices A_i and the opposite of each cone can be precomputed offline. The naive algorithm works in $O(n_f)$. Intuitively, Alg. 2 roughly follows a one-dimensional sequence of cones within a two-dimensional arrangement of cones, so that it should perform in $O(\sqrt{n_f})$. This is exactly what we observe empirically (see next section). It would be possible to devise scheme in $O(\log n_f)$ by implementing a tree of tests where each leaf contains only one cone, be it based on the cones' faces or on an octree/spherical quadtree, but (i) our scheme is easy to implement and fast enough (only taking a few percents of the total ray-casting algorithm), (ii) it can take advantage of spatial coherency, *i.e* when a request is made with directions d close to the previous request (as is often the case in our application cases), one can use the preceding solution to initialize the algorithm. For small changes in d we can reach amortized $O(1)$.

VII. Tests and examples

A. Timings

We illustrate here the speed of our algorithm. All computations are performed on a Intel Core i7-4960 HQ at 2.6GHz, on a single core.

We begin with measurements for Alg. 2. For objects with increasing number of faces, we first test for a large number of directions d, uniformly distributed over S^2, initializing the algorithm with $i_0 = 0$. Then we do the same, but for each d we make 1000 tests, using $d_k = R(k\theta)d$ ($k = 1..1000$) as a direction where R denotes a rotation of angle $k\theta$ around an axis orthogonal to d, and θ is an angle step that we take from 0.001 to 0.5. For these tests, we take i_0 as the solution

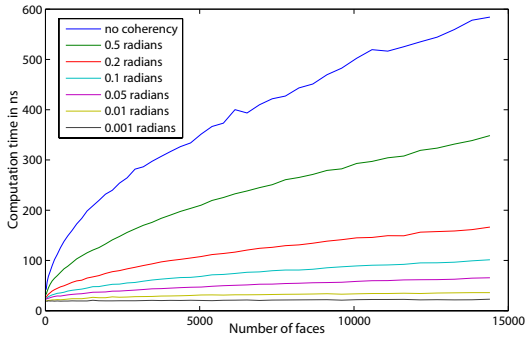

Fig. 7. Computation time (in ns) for one call to Alg. 2 w.r.t. the number of faces of the control mesh. The curve in blue shows the case where no specific initialization was made. The others illustrate the use of spatial coherency for directions closer and closer from one another.

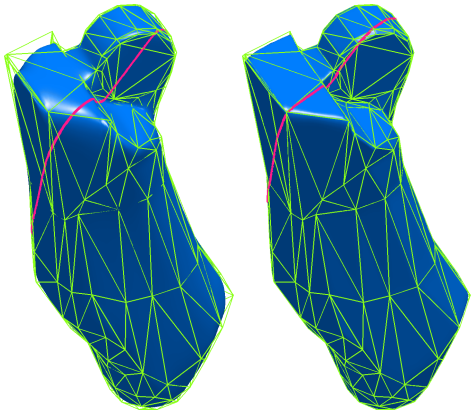

Fig. 8. Left: the CCS (in blue) corresponding to the mesh (in green) of an HRP-2 link. The red line is the image of $R(k\theta)d$ for a particular d. Right: we first apply twice a simple subdivision scheme, before using Catmull-Clark subdivisions, thus obtaining a surface closer to the original mesh but less smooth. The red line corresponds to the same directions as on the left, but is slightly changed due to a difference in the center c.

of the previous request. The results are depicted in Fig. 7. They clearly show the $O(\sqrt{n_f})$ behavior when no spatial coherency is used, and a quasi-constant computation time for the lower increments θ.

It is difficult to give a systematic and fair timing of Alg. 1, because it is depends much more on how close the CCS is from the control mesh (and hence how good the initial guess is for Newton iterations) than on number of faces. It is not easy to measure this proximity and much less to generate families of objects with increasing complexity w.r.t. this measure. An extreme example about the influence of the number of faces is to take a cube and divide each of its faces in k by k, $k = 1...$ to generate a family of control meshes. In this case the computation time decreases with the number of faces, with the average number of Newton iterations even dropping below 1 when faces are divided in 6 by 6 or more! While at first counter-intuitive, this is not a surprise: the more the faces are divided, the closer the CCS is to the control mesh. In the case of the cube, more and more patches are equal to their corresponding face. This behavior is actually general. It suggests than performing explicitly a few more Catmull-Clark iterations on the control mesh might actually improve the algorithm speed, until Alg. 2 becomes limiting. We keep the study of this as future work.

To still get an idea of the speed of the algorithm we perform a test similar to the coherency test above, with $\theta = 2\pi/1000$, on an object typical of our target applications: a leg body of the HRP-2 robot. The result for one particular d is depicted in Fig. 8. In average, the computation time for one call to Alg. 1 is $5.3\mu s$, 3.2 Newton iterations are performed, and there is 0.048 switches to a neighbour face.

B. Posture generation examples

We now show some examples of PG using the proposed approach to introduce complex objects to contact with. We make use of the capacity of our optimization solver (see [5]) to directly work with variables on S^2 to represent the direction d. When using more classical non-linear solvers, it is sufficient to take d in $\mathbb{R}^3$ and to add a constraints of unit norm for d: the presented algorithm do not need a unit d to work, and at the solution, the gradient will be full rank on the null-space of this constraint, which is enough for a good convergence. However, working directly on S^2 presents advantages [5].

In the first example (Fig. 9, left), a HRP-4 robot is asked to grasp an object (here a leg part of HRP-2) and to hold it as far as possible in front of him, with the following constraints: contacts must occur between predefined points on the hands of HRP-4 and points free to move on the surface of the object, modeled as a parametric surface with the approach of this paper; furthermore the robot has to keep its left foot at a given fixed position and has its right foot anywhere on the floor. The forces applied by the robot on the object must be sufficient to counter the gravity, and the system (robot,object) must be stable with all forces within their friction cones. Additionally, the robot must stay within its joint limits. We do not check here for collisions or torque limits. The solver finds a solution in about 70 iterations.

Our second example (Fig. 9, middle) demonstrates our PG applied to grasping an object. The base of the hand is fixed, and the contact points on the fingers are given. The contact points on the objects are parametrized with our method and are free to move on the entire (approximated) surface of the object: the optimization finds automatically how to grasp the object so as to be able to maintain it, taking about 50 iterations. Note that the object is non-convex.

Our third example (Fig. 9, right) involves a stack of cubes on which the robot must stand, using its hands and feet to maintain its stability. The stack is modeled as a single object. Once again, contact points are fixed on the robot and free on the surface. The optimization takes around 115 iteration to converge to a solution. Since their is a single surface to contact with, we do not need to specify with which faces, edges or corners the robot needs to be in contact with. The optimization decides it automatically. This is a very attractive feature of our approach as it allows to include discrete choices directly into our PG. This can be used to alleviate the work to be done by the user, be it a human or a planning algorithm: the user still has to specify the bodies in contact, but part of the combinatorics relative to the matching of a body with a particular surface or object is

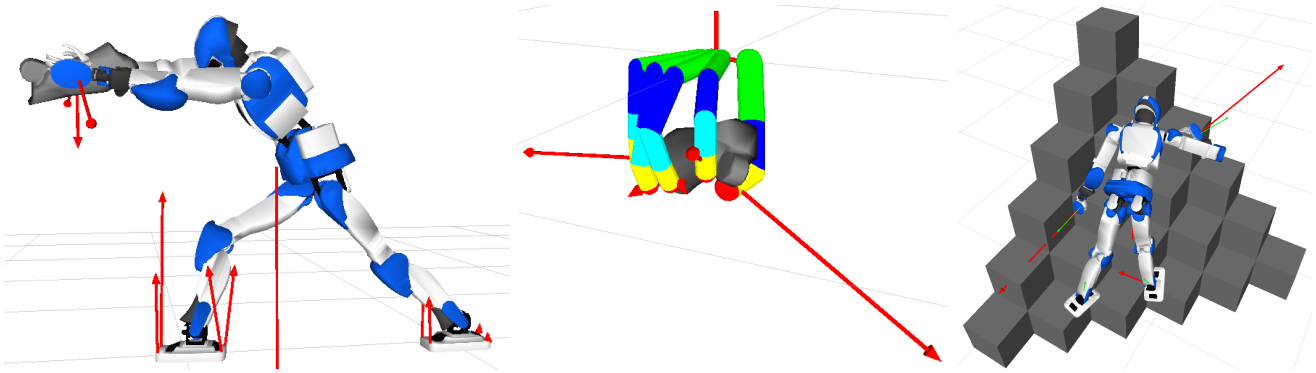

Fig. 9. Left: HRP-4 holding an object, modeled with CCS, as far as possible in front of it. The red arrows depict the contact forces and the gravity forces applied at each object. Middle: example of grasp generation. Right: HRP-4 climbing a pile of cubes modeled as a single object with a single surface.

handled directly by the PG. Note that here again we do not check for collision, so that we have a slight penetration of the left foot with the cube in front.

VIII. Conclusion

In this paper, we presented a method to parametrize the surface of closed objects so that they can be used in optimization problems. We validated the approach through PG examples.

We use CCS to smooth meshes, but our scheme needs actually only patches of parametric surface to work with, so that it could be extended to many other surface subdivision techniques, or patches of NURBS for example. This would allow for possibly more continuity properties.

The proposed parametrization is C^1 everywhere and even C^2 but at extraordinary vertices. Normal vectors to the parametrized surface are C^1 everywhere but at extraordinary vertices. Although this violates the requirements for optimization methods, it happens at isolated points and is not a problem: the probability to converge to such a point is 0, and even if close to those points the gradient becomes large, this is not an issue in practice, although not fully ideal.

Depending on the control mesh, the CCS may not be a close enough approximation. However we can make the approximation as tight as we want by simply subdividing the edges and faces with their center points and keeping the original vertices, which corresponds to taking weights $\alpha_e = 1$, $\beta_e = 0$, $\alpha_v = \beta_v = 0$ and $\gamma_v = 1$ in the subdivision scheme of Sec. II. Indeed, bi-cubic B-splines have the very important property to be contained in the convex hull of their control points. Therefore, adding points on the original mesh will constrain the patches to be closer to the control mesh. An illustration of this is depicted in Fig. 8 (right). Another option would be to use extensions of the CCS, especially the notion of *crease* that allows for sharper edges and corners while retaining smoothness. Of course, the closer we become to the original mesh, the closer we are of loosing gradient continuity at edges and vertices.

We took here the point of view of approximating a mesh by a CCS. However in computer graphics, this is usually the other way around: the control mesh is chosen so that the CCS match the intended object, an approach that we can use too and that would eliminate the possible problem of approximation.

The ability to take contacts anywhere on a complex surface is very attractive. Although we did not show it here, parametric surfaces can also be used to have non-fixed contact point on the robot. However, if contact is made between points of two parametric surfaces, it requires to be able to associate a full frame to one of the points, not only a normal, so as to write qualified constraints, which is not always possible.

This ability of contacting anywhere also raises the problem of collisions, when faced with non convex surfaces, what we purposefully oversaw in this paper. Being able to write at the same time that a body must be in contact with a surface and not collide with it is not a simple task and in particular requires a careful handling of safety margins. This is one of our top priority in extending the capabilities of our PG.

References

[1] K. Hauser, T. Bretl, and J. C. Latombe, "Non-gaited humanoid locomotion planning," in *IEEE Intl. Conf. of Humanoid Robots*, 2005.

[2] A. Escande, A. Kheddar, and S. Miossec, "Planning support contact-points for humanoid robots and experiments on HRP-2," in *IEEE/RSJ International Conference on Robots and Intelligent Systems*, Beijing, China, 9-15 October 2006, pp. 2974–2979.

[3] J. Vaillant, A. Kheddar, H. Audren, F. Keith, S. Brossette, K. Kaneko, M. Morisawa, E. Yoshida, and F. Kanehiro, "Vertical Ladder Climbing by HRP-2 Humanoid Robot," in *IEEE-RAS Int. Conf. Humanoid Robot.*, Madrid, Spain, 2014, pp. 671–676.

[4] A. Escande, A. Kheddar, and S. Miossec, "Planning contact points for humanoid robots," *Robotics and Autonomous Systems*, vol. 61, no. 5, pp. 428–442, May 2013.

[5] S. Brossette, A. Escande, G. Duchemin, B. Chrétien, and A. Kheddar, "Humanoid posture generation on non-euclidean manifolds," in *15th IEEE RAS Humanoids Conference*, 2015, to appear.

[6] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[7] J. Stam, "Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '98. New York, NY, USA: ACM, 1998, pp. 395–404.

[8] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[9] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[10] W. Martin, E. Cohen, R. Fish, and P. Shirley, "Practical ray tracing of trimmed nurbs surfaces," *Journal of Graphics Tools*, vol. 5, no. 1, pp. 27–52, Jan. 2000.

[11] D. Eberly, "Perspective mappings," Geometric Tools, LLC, Tech. Rep., August 2011.

[12] U. Reif, "A unified approach to subdivision algorithms near extraordinary vertices," *Computer Aided Geometric Design*, vol. 12, no. 2, pp. 153–174, Mar. 1995.