

---

# 3D-graphics for iterated integrals

Shannon Holland\*

Matthias Kowski<sup>†‡</sup>

---

**Abstract:** Finding the limits of integration for iterated integrals is one of the hardest and most error-prone tasks in multi-variable calculus. This article introduces two MAPLE programs for visualizing the corresponding parameterizations of planar and 3D-regions. The programs graphically validate limits, and provide graphic feedback how to correct mistakes. The instant visual feedback empowers students to work on their own, and provides relief for graders and instructors. Aesthetic aspects play an important role for motivation. The programs are designed to be extremely user-friendly, yet support a variety of sophisticated options.

## Double and triple integrals

Double and triple integrals are a core topic in the multi-variable calculus curriculum. Using Fubini's theorem, multi-dimensional integrals may be reduced to iterated integrals. Students unequivocally consider setting up these integrals, that is, finding the limits of integration, one of the hardest tasks in the course. This really is not a calculus-topic at all, but rather is a question of describing a region using inequalities in the special format:

$$\mathcal{R} = \{(u, v, w) : a \leq u \leq b, f(u) \leq v \leq g(u), h(u, v) \leq w \leq k(u, v)\} \quad (1)$$

Depending on the integrand and the shape of the region, it may be beneficial to change the order of the variables, or use a different coordinate system, most commonly polar, cylindrical, or spherical coordinates. (Throughout, the article uses  $u, v, w$  which may stand for any of these coordinates.) Frequently it is necessary to break a region up into several simple pieces, each of which may be parameterized by a single expression of the form (1). The composite region may not allow any single such description.

The problem of setting up iterated integrals is exacerbated by the almost complete lack of facilities to check the answers: Text-books, graders, instructors and fellow students rarely can provide enough timely reliable feedback. An advanced technique checks correctness by comparing numerical evaluations using *generic* integrands

---

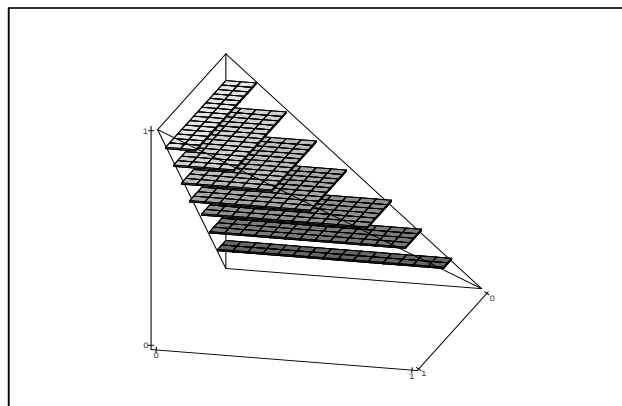
\*Center for Innovation in Engineering Education, Arizona State University, Tempe, AZ 85287

†Department of Mathematics, Arizona State University, Tempe, AZ 85287

‡This work was supported by the NSF through the Foundation Coalition under Cooperative Agreement EEC 92-21460.

and a set of different orders of the variables, or different sets of coordinates. This technique is usually not considered appropriate for sophomores.

Many students have difficulties visualizing sliced regions in 3-space. To recognize how difficult it is to visualize the slices (parallel to the coordinate planes) may be for a skew triangular pyramid inside the unit cube:



```
> intdraw3d(x=0..1,y=1-x..1,z=x..1,  
> grid=[9,17,17],spaces=[.95,0,0]);
```

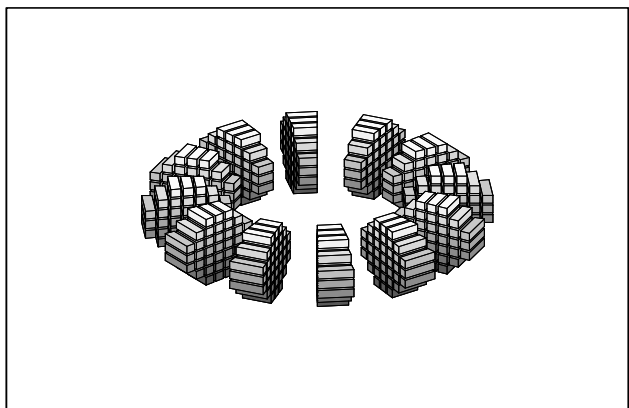
Depending on the choice of vertices, few students believe that such cross-sections can be rectangles, rather than just triangles. The picture shows the relatively easy case of vertices at  $(0, 0, 0)$ ,  $(0, 1, 0)$ ,  $(0, 0, 1)$ , and  $(1, 0, 1)$ .

## The programs `intdraw`/`intdraw3d`

Once a multiple integral is written as an iterated integral, the symbolic evaluation is relatively straightforward (if at all possible). Numerical evaluation poses some interesting problems for complicated regions, but these are adequately treated in the classical literature, e.g. [1]. Our program only address “*setting up the iterated integral*”. It does not automate finding the symbolic limits. Rather it is meant to be a facility to visualize and graphically validate any symbolic solution. A program that would automatically set up iterated integrals would necessarily be limited to very specific classes of regions, and even then would remain a formidable challenge to write. Even more important for educational use is that such a program would do little to improve the visualization skills, and to help fix mistakes.

Graphical information can be very compelling feedback: Students incessantly, iteratively improve their so-

lutions and gain command of the topic. Successful examples are the programs *FINDPOLY*, *TWIDDLE* and *IDENTIFY FUNCTION* of the *ARIZONA MATHEMATICAL SOFTWARE* [2] which ask students to find formulas given graphs of functions. (These belong to a suite of “freeware” that accompanies the Harvard Consortium Core Calculus Curriculum.) This principle of providing instantaneous graphical feedback as a powerful motivation to work incessantly *until the picture*, i.e. the solution, *is correct*, has been most successfully employed by the second author in various class projects.



```
> intdraw3d(theta=0..2*Pi,z=-1..1,
>   r=3-sqrt(1-z^2)..3+sqrt(1-z^2),
>   grid=[12,7,7],spaces=[.6,.2,.2]);
```

Our programs take as input the four or six limits of integration in any of the common coordinate systems (rectangular, polar, cylindrical and spherical), in any order of the variables, and display the corresponding **sliced** regions as output.

It is essential to not simply display the solid region described by (1), as typically, at least initially, students’ solutions contain some errors: The corresponding solid regions usually provide little specific feedback of which limits are wrong. Moreover, displaying solid regions does not help to visualize the different ways of slicing that correspond to different orderings of variables (or different sets of coordinates). The similar programs *dxdyplot*, *drdtplot*, etc. of [3] recognize this need and provide a graphical clue for the slicing in two dimensions, but the three-dimensional versions *dxdydzplot*, etc. apparently lack this crucial facility.

Our experience shows that it suffices to consider relatively simple, common shapes when learning to set up iterated integrals: This is crucial, as for checking the correctness of any limits of integration with our programs, the user should already have a mental image of the region. Commonly we use familiar objects in 3-space like rectangular and skew pyramids, portions of cones,

spheres and cylinders, torii, etc. which every student recognizes readily. However, the program may also be used to visualize regions that are already given in the special form (1). It is also a powerful tool for gaining a pictorial understanding of the slices corresponding to a different ordering of variables, e.g. when the task is to change the order of integration. Compare the section on “Slices and projections” for a description of how to manipulate the optional argument `spaces=[...]` for such a purpose.

The programs are extremely user-friendly. The input consists of the (two or) three ranges of integration:

```
> intdraw(u=u0..u1,v=v0..v1);
> intdraw3d(u=u0..u1,v=v0..v1,w=w0..w1);
```

The program accepts only the variables  $x, y, z, r, \Theta, \varrho, \Phi$ , and automatically recognizes rectangular, polar, cylindrical and spherical coordinates, e.g. by the presence of  $x, r$ , or  $\Phi$ . While this conflicts with standard plot-functions in MAPLE, it saves much typing, and was strongly requested during in-class testing. We justify this by observing that the program will be used almost exclusively in a limited setting where little confusion may arise. The only ambiguity arises from the ordering of the three ranges: We chose to use the same order of the limits as in the standard notation

$$\int_{u=u_0}^{u_1} \int_{v=v_0}^{v_1} \int_{w=w_0}^{w_1} f(u, v, w) dw dv du$$

This order was preferred by students who tested the program. However, it conflicts with the ordering needed for evaluation of iterated integral using MAPLE:

```
> int(int(int(f,w=w0..w1),v=v0..v1),u=u0..u1);
```

The program performs significant error- and type-checking up-front and generates specific error messages for various common mistakes. E.g. it checks whether the variables are from the set of admissible variables, and whether the limits are of the right type: The outside limits  $u_0$  and  $u_1$  must be numerical constants with  $u_0 < u_1$ ,  $v_0$  and  $v_1$  may be expressions depending on  $u$ , and  $w_0$  and  $w_1$  may be expressions depending on  $u$  and  $v$ . The program is flexible enough to accommodate e.g. cases where  $f(u) \leq g(u)$  does not hold for all  $u \in [u_0, u_1]$ , and then displays a region with the obvious crossing of the boundaries.

With such ease of use, students only need minimal instruction (if at all), and primarily use the program at home or on their own in open labs. We found it helpful to encourage work in teams, which often leads to intense discussions of which limit is wrong, why, and how it might be fixed. The program helps students learn to visualize sliced regions in space. Arguably, the main achievement is that it empowers students to work on

their own: they can work many more problems, and check answers on their own as opposed to being restricted to the few problems with answers in the back of the book, having to wait for the grader to return the homework, or wait for the instructor. Indeed, they now commonly try to set up integrals in all possible orders, often trying different coordinates as well. As a side benefit, the instructor saves much time as students check their own work, neither turning in false answers, nor asking the instructor for corrections.

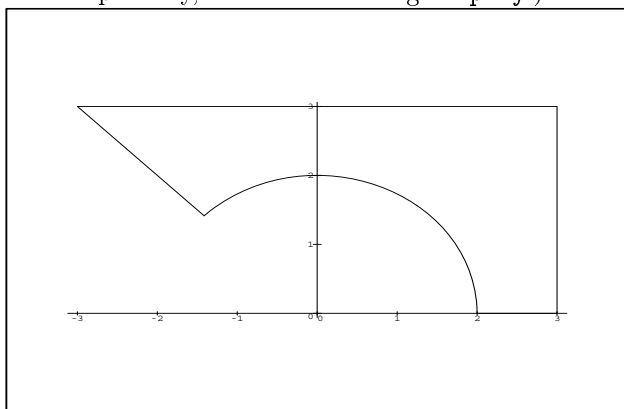
The programs make relatively little use of the symbolic computer-algebra capabilities of MAPLE, and could in principle have been written in various other languages. However, we found it important to use the same package that is used daily in the classes. In particular, setting up integrals is typically followed by evaluating the iterated integrals – and MAPLE is the package of choice for integration in closed form.

## Catching a typical mistake

Even experienced mathematicians often make mistakes when setting up iterated integrals. Initially, an object may be described by a set of unstructured inequalities (corresponding to the bounding surfaces). Expressions in the special form (1) are obtained by combining inequalities, and / or solving them for individual variables. The main difficulties arise from having to pick one of multiple solutions. A simple planar example is

$$\mathcal{R} = \{(x, y) : -y \leq x \leq 3, 0 \leq y \leq 3, x^2 + y^2 \geq 4\}$$

The following picture shows an outline of the region. (The programs `intdraw/intdraw3d` do not supply outlines of the regions. Those shown in this article are generated separately, and overlaid using `display`.)



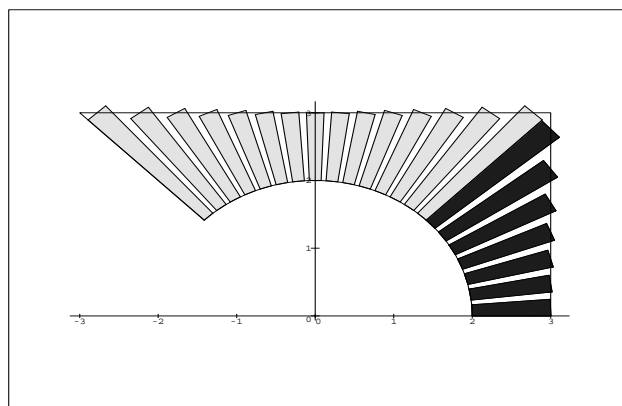
```
> c:=display({plot([2*cos(t),2*sin(t),t=0..3*Pi/4],
>   plot([[sqrt(2),sqrt(2)],[-3,3],[3,3],[3,0],
>     [2,0]])}, color=black,thickness=2,
>   scaling=constrained,tickmarks=[5,3]):
> c;
```

Depending on the integrand one may decide to set up an

iterated integral in polar coordinates. One rewrites each inequality in polar coordinates, and solves for the variable of the inside integral as a function of the variable of the outside integral. The limits of the outside integral are obtained by solving pairs of equations. Rather than using piecewise defined functions in the limits, it is common practice to split up the integral.

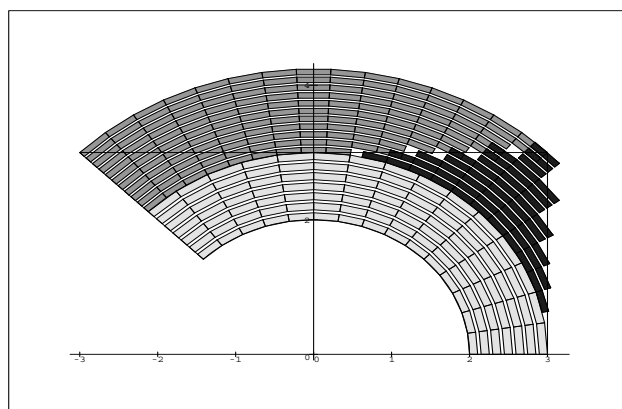
$$\int_0^{\frac{\pi}{4}} \int_2^{\frac{3}{\cos \theta}} f(r, \theta) r \, dr \, d\theta + \int_{\frac{\pi}{4}}^{\frac{3\pi}{4}} \int_2^{\frac{3}{\sin \theta}} f(r, \theta) r \, dr \, d\theta$$

A call of `intdraw` visually validates the new limits



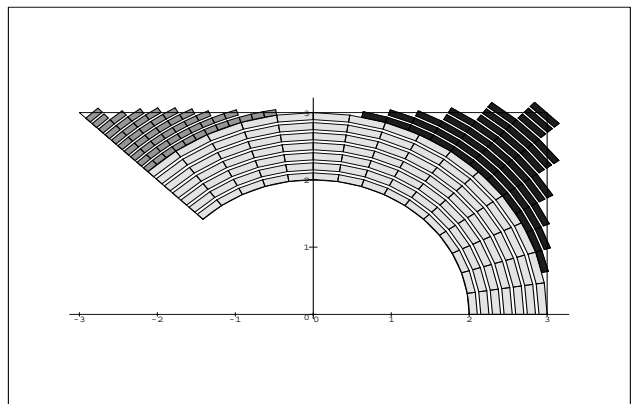
```
> r1:=intdraw(theta=0..Pi/4,r=2..3/cos(theta),
>   color=blue,grid=7,title='',style=patch):
> r2:=intdraw(theta=Pi/4..3*Pi/4,r=2..3/sin(theta),
>   color=yellow,grid=15):
> display({c,r1,r2});
```

For the reversed order of integration, a common mistake results from incorrectly solving  $y = 3$  for  $\theta$  in terms of  $r$ , giving  $\theta = \sin^{-1}(3/r)$ . A call of `intdraw` yields:



```
> c1:=intdraw(r=2..3,theta=0..3*Pi/4,
>   color=yellow,grid=7):
> c2:=intdraw(r=3..3*sqrt(2),theta=arccos(3/r)
>   ..arcsin(3/r),color=blue,grid=11):
> c3:=intdraw(r=3..3*sqrt(2),theta=arcsin(3/r)
>   ..3*Pi/4,color=green,grid=11):
> display({c,c1,c2,c3});
```

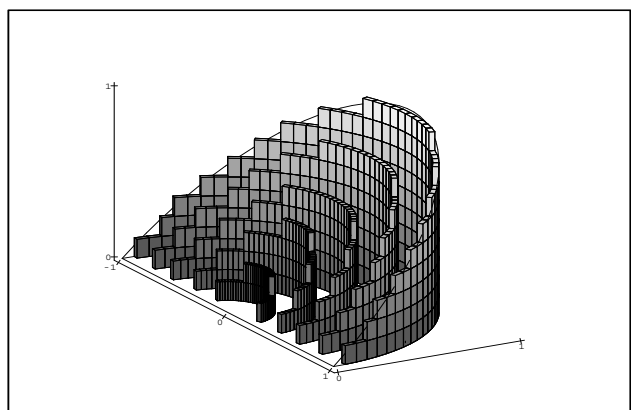
The picture clearly demonstrates the mistake. Two of the regions are parameterized correctly. For the third (green, or medium-gray) region, the lower and upper limits for  $r$  appear correct, but the lower limit for  $\Theta$  is false. This is easily traced back to the equation  $y = 3$ : One needs to pick *the other* solution  $\Theta = \pi - \sin^{-1}(\frac{3}{r})$ . This mistake can be avoided by working with inequalities throughout, but most people prefer to manipulate equations. A call to `intdraw` confirms the correct solution.



```
> c32:=intdraw(r=3..3*sqrt(2),theta=Pi-arcsin(3/r)
>           .3*Pi/4,color=green,grid=11):
> display({c,c1,c2,c32});
```

## Slices and projections

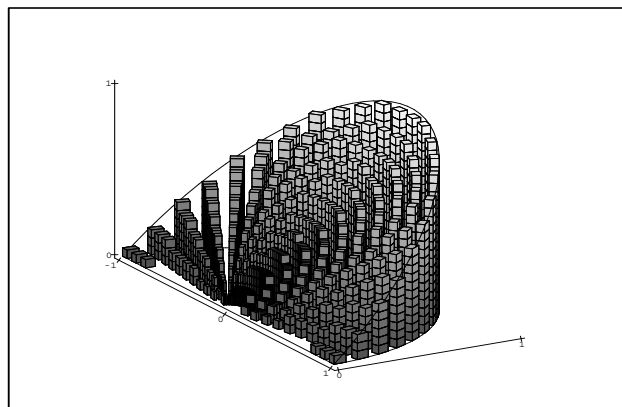
For three dimensional regions there are two principal points of view: Grouping the inner two integrals together corresponds to considering slices with the outer variable  $u$  held constant. The picture shows the “wedge  $\{(x, y, z) : 0 \leq z \leq y\}$  cut out of the tree”  $\{(x, y, z) : x^2 + y^2 \leq 1\}$ , sliced along  $u = r = \text{const.}$



```
> intdraw3d(r=0..1,theta=0..Pi,z=0..r*sin(theta),
>           grid=[6,48,9],spaces=[.85,0,0]);
```

Alternatively, grouping the outer integrals together cor-

responds to projections along the curves  $u = \text{const.}$ ,  $v = \text{const.}$  onto a surface  $w = \text{const.}$  In the rectangular case, this is the projection parallel to the  $w$ -axis onto the  $uv$ -plane. The picture shows columns aligned with the  $w = z$ -axis *standing on the shadow* in the  $xy = r\Theta$ -plane.



```
> intdraw3d(r=0..1,theta=0..Pi,z=0..r*sin(theta),
>           grid=[6,48,6],spaces=[.5,.5,0]);
```

Most popular textbooks and many instructors emphasize the projection-point-of-view. Nonetheless, for different regions, either point of view may be more appropriate. The slicing appears to be particularly useful when dealing with curvilinear coordinates.

The program `intdraw3d` supports both points of views. It internally generates a three dimensional grid, and draws boxes (centered at these grid points) whose sides are aligned with the surfaces  $u = \text{const.}$ ,  $v = \text{const.}$ , and  $w = \text{const.}$  The respective impressions of columns standing on a shadow or thin slices that themselves are sliced into strips are controlled by the amount of open space between the boxes in the respective directions.

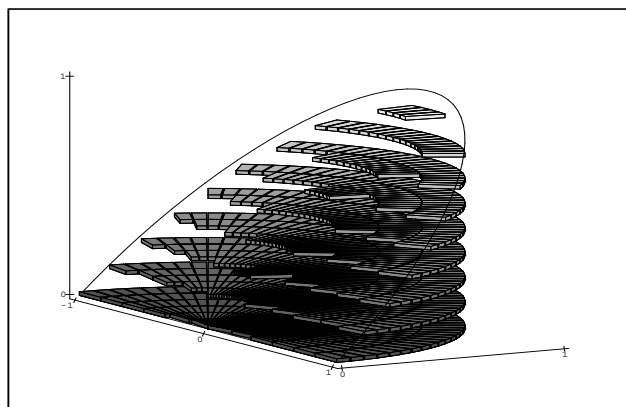
The spacing may be controlled with the optional argument `spaces=[pu,pv,pw]`. Each parameter `px` must lie in the interval  $[0, 1]$ , and determines the open space between the boxes as a fraction of the total space in that direction. The default value is `spaces=[.3,.3,0]`, which gives the impression of columns standing on a shadow. Also preprogrammed is the choice `spaces=slices` which is equivalent to `spaces=[.8,.1,0]`. Missing third arguments are interpreted as zero, i.e. no visible slicing in the third direction. The `spaces` parameter invites some particularly nice animations: For example,

```
> display([seq(intdraw3d( ...,spaces=[(8-k)/10,
>           (1+k)/10,0]),k=0..7)],insequence=true);
```

generates an animation of the object that corresponds to changing the order of integration of the outer two variables. Such animations may take a few minutes to compute and render (in particular when using fine grids). Yet, the display can be truly stunning. Feedback by stu-

dents has been overwhelmingly positive, claiming that it much helps improve their 3D-visualization skills.

As a corollary, we note that this trick may be also used to get insight as to what the slices corresponding to reordered variables look like: Suppose one knows one description of a solid region in the form (1). Use `intdraw3d` with the given limits, but use a spacing that forces the display to correspond to a different order of slicing! For example, using exactly the same limits as above, but forcing the spaces between the boxes to correspond to an apparent different order of variables shows what the slices in a horizontally cut wedge look like.



```
> intdraw3d(r=0..1,theta=0..Pi,z=0..r*sin(theta),
> grid=[6,48,9],spaces=[0,.1,.85]);
```

## Some remarks on the grid

The procedures `intdraw` and `intdraw3d` accept practically all standard `plot` or `plot3d` options, like `color`, `light`, `style`, `orientation`, etc. These will be directly passed through to the output. The optional parameter `spaces` has been discussed above. An additional optional parameter is `grid=g`, which controls the number of slices, columns, or boxes drawn (and thus is different from its usual role as an optional argument).

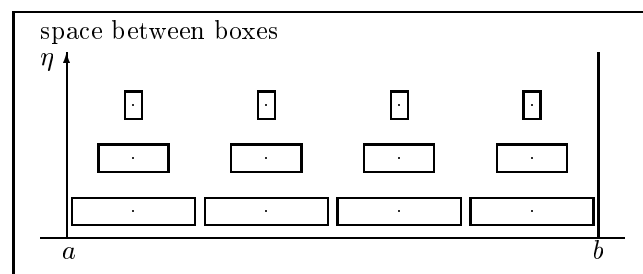
For `intdraw` the specification `g` may either be a positive integer or a list of two positive integers, for `intdraw3d` it may be a list of two or three positive integers. If `g` is a list of two integers in `intdraw3d` (or an integer in `intdraw`) then the “lengths” of the columns are  $k(u_i, v_j) - h(u_i, v_j)$  (or  $g(u_i) - f(u_i)$ ) (see below for the special case when the innermost variable is  $\Theta$  or  $\Phi$ ). This is the preferred option. If `g` is a list of three integers in `intdraw3d` (or a list of two integers in `intdraw`) then only the respective number of *quantized* column lengths is available – using a small value for this third (second) parameter may be confusing for beginners.

If no grid is specified the default `grid=[5,5]` (`grid=15` in `intdraw`) is used. If the third (second in `intdraw`) variable is either  $\Theta$  or  $\Phi$ , the “columns” are

curved, and the default is `grid=[5,5,5]` (`grid=[15,15]` in `intdraw`). Finer grids require substantial more time for computations and rendering. For the purpose of checking limits in iterated integrals quite crude images often suffice!

The experiences and rationale behind some choices of the grid are noteworthy. A preliminary version of the program `intdraw3d` drew  $m$  two-dimensional slices that themselves were each sliced into always the same number  $n$  of rectangles, disregarding the width of each slice. In-class experiences suggested that rather than using a fixed number of strips for each slice, the strips should have uniform width. Also, instead of two-dimensional rectangles, three-dimensional boxes were requested by students. Even though each of these has six rectangles as sides, the overall computing time did not increase much as a considerably smaller number of rectangular boxes gives as useful a picture as a larger number of two-dimensional rectangles.

The next pilot version of the program generated a grid of midpoints of the rectangular boxes that was uniform in each direction. If  $u=a..b$ ,  $v=f(u)..g(u)$  and `grid=[m,n]`, it defined  $u_i = a + (2i - 1)/2 \cdot (b - a)/m$ . With  $c = \min_i h(u_i)$  and  $d = \max_i k(u_i)$ , it used  $v_j = c + (2j - 1)/2 \cdot (d - c)/n$ . For each pair  $(i, j)$  it checked whether  $f(u_i) \leq v_j \leq g(u_i)$ , and used this to find the maximal and minimal  $w$ -value, and thus the coordinates  $w_k$ . Finally, for each point  $(u_i, v_j, w_k)$  that lies inside the region a box with this mid-point is drawn, with dimensions specified by the `spaces` option.

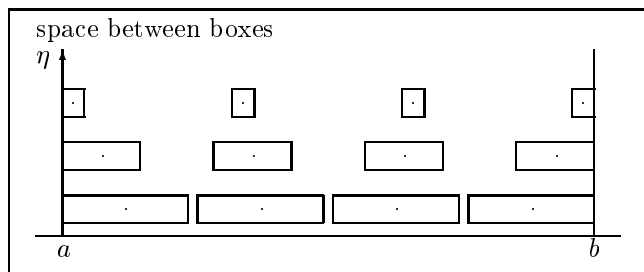


Boxes with fixed grid of center points

This grid of midpoints, independent of the `spaces` option had some very undesirable effects. For example if the spacing was chosen so that there was a large open space between the boxes in the  $u$ -direction, then the left edge of the first box at  $u_1 - (1 - \eta) \cdot \Delta u/2$  could be quite far away from the left end  $u = a$  of the region. In particular, for a cone the resulting picture was quite unappealing when (with  $u = \varrho$ ) the *first* slice did not start at the vertex.

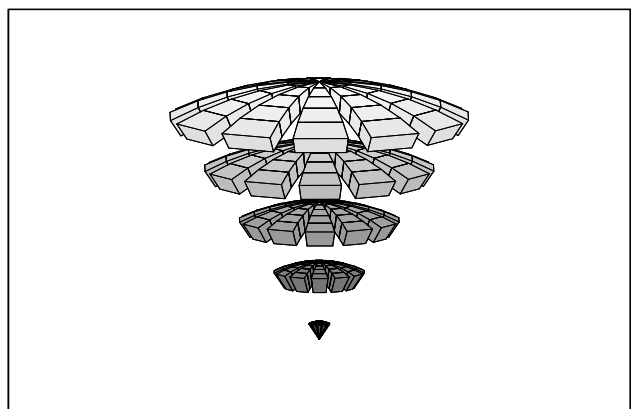
The final version presented here uses a grid that adapts to the open space between the boxes. Specifically,

suppose `spaces=[eta,...]`, `grid=[n,...]`, `u=a..b` as before. Set  $\Delta u = (b - a)/(n - \eta)$  and  $u_i = a + (i + (1 - \eta)/2)\Delta u$  for  $i = 1, \dots, n$ . Each box centered at  $u_i$  again has width  $(1 - \eta)\Delta u$ . This way, each of the  $(n - 1)$  open spaces between the  $n$  boxes in this direction is still  $\eta$  times the distance between any two midpoints, while now the left edge of the first box is always at  $u = a$  etc.



Boxes with moving grid of center points

In an analogous way the grid of midpoints accommodates the extreme values in the  $v$  and  $w$ -directions. The additional computational effort is minimal. One may observe the shifting of the boxes when viewing an animation that changes the `spaces` parameters. Some users are surprised to see shifting boxes in animations, but feedback indicates that the extra work is well worth the effort: After all, when using the picture to check limits of integration, one typically first looks for the extreme values – and it can be quite irritating if the object does not extend all the way to the expected *corners*.

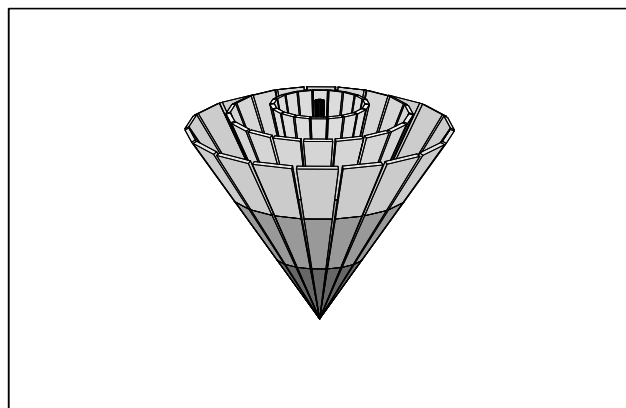


```
> intdraw3d(rho=0..1,theta=0..2*Pi,phi=0..Pi/6,
> axes=None,grid=[5,12,5],spaces=[.7,.3,0])
```

One possible further improvement could be to replace the *flat* bottoms and tops of the columns drawn by `intdraw3d` by tilted polygons. For *columns* in the interior this is quite easily done by evaluating e.g. the limit  $w = h(u, v)$  at several corners  $(u, v) = (u_i \pm \Delta u, v_j \pm \Delta v_j)$  instead of at  $(u, v) = (u_i, v_j)$  (here  $2\Delta u$  is the width of

the box in the  $u$ -direction etc.) However, then one needs to again check whether the corners still lie in the domain, e.g. whether  $h(u_i \pm \Delta u, v_j \pm \Delta v_j) \leq k(u_i \pm \Delta u, v_j \pm \Delta v_j)$ . (Along the edges additional problems may arise as the new corners may not lie inside the domain of  $h$  and  $k$  etc., even if only due to roundoff-errors.) Alternatively, one might want to use symbolically computed derivative information for the tilt.

By the same reasoning, one might also request that instead of drawing columns with rectangular cross-sections, any columns at the boundary should have polygonal cross-sections approximating the boundary. We decided that the additional work would not be worth the efforts, as for the main use, checking limits of iterated integrals, the rectangular columns are satisfactory.



```
> intdraw3d(rho=0..1,theta=0..2*Pi,phi=0..Pi/6,
> axes=None,grid=[5,12,5],spaces=[.7,.3,0])
```

**Software:** The MAPLE programs `intdraw` and `intdraw3d` are part of the package `asu.ms`. This, and related information and programs, are available on the WorldWideWeb-site of the second author <http://math.la.asu.edu/~kawski/maple.html>, or by anonymous ftp from `math.la.asu.edu` in the directory `pub/kawski/MAPLE`. A *HELP*-utility in standard MAPLE format and information on how to install the package are included.

**Acknowledgments:** The authors are very thankful for the generous support by the Foundation Coalition. The second author gratefully acknowledges the feedback from his students.

## References

- [1] A. H. Stroud, *Approximate Integration of Multiple Integrals*. Prentice-Hall, 1971.

- [2] D. Lomen and D. Lovelock, *The ARIZONA MATHEMATICAL SOFTWARE*, <http://www.math.arizona.edu/software/uasft.html>.
- [3] C. K. Cheung and J. Harer, *Multivariable Calculus with MAPLE V* (preliminary version) Wiley, 1994.

**The authors:**

**Shannon Holland** wrote most of the package as a freshman in the Integrated Engineering Program of the Foundation Coalition at Arizona State University. He now is a junior in computer systems engineering, and is very interested in 3D-visualization, using packages like 3D-STUDIO MAX and exploring the capabilities of JAVA and VRML.

e-mail address: [sk.holland@asu.edu](mailto:sk.holland@asu.edu)

**Matthias Kawski** attended the Universities of Hamburg, Tübingen, and Colorado. He received his Ph.D. in 1986 in Boulder, CO, and joined the faculty of Arizona State University in 1987. He works in differential geometric control theory, recently with excursions into algebraic combinatorics, and regularly uses MAPLE in his research. He has been very active in calculus reform and curriculum integration, constantly exploring new uses of MAPLE at the introductory level.

URL-address: <http://math.la.asu.edu/~kawski>